НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій Стіренко

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Система автоматичного оцінювання творів»

Виконав:
студент IV курсу, групи ІО-64
Фельдман Михайло Георгійович _____

Керівник:
Старший викладач
Саверченко Василь Григорович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Sen. Java Dev., DXC Luxoft, к.т.н.,
Невдащенко М. В. _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Сергій Стіренко

«___»_____2020 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Фельдману Михайлу Георгійовичу

1. Тема проєкту «Система автоматичного оцінювання творів», керівник проєкту Саверченко Василь Григорович, старший викладач, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту *26 травня 2020р.*

3. Вихідні дані до проєкту *див. технічне завдання*

4. Зміст пояснювальної записки *Аналіз і характеристика об'єкта проектування, обґрунтування оптимального варіанта реалізації мети цієї роботи, розробка додатку: вибір технологій та їх обґрунтування, основні рішення з реалізації додатку. Висновки.*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) *приниципова схема, функціональна схема, структурна схема*

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Сімоненко В. П., професор, д.т.н.		

7. Дата видачі завдання 01.09.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми роботи*	*01.09.2019*	
2.	*Вивчення та аналіз завдання*	*15.12.2019-15.03.2020*	
3.	*Розробка архітектури додатку*	*15.03.2020-25.03.2020*	
4.	*Написання програмної частини*	*25.03.2020-05.04.2020*	
5.	*Тестування та виправлення помилок*	*05.04.2020-15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04.2020-20.05.2020*	
7.	*Захист програмного продукту*	*25.04.2020*	
8.	*Передзахист*	*26.05.2020*	
9.	*Захист*	*18.06.2020*	

Студент Михайло Фельдман

Керівник Василь Саверченко

Анотація

Дипломна робота присвячена вирішенню завдання автоматизованого оцінювання невеликих за обсягом прозових творів (есе) із використанням методів та засобів комп'ютерної обробки природної мови (англ. natural-language processing, NLP). У зв’язку з тим, що останнім часом інтерес до самоосвіти та масових відкритих онлайн-курсів (англ. massive open online course) постійно зростає, системи, які дозволяють автоматизувати перевірку академічної успішності, набувають великої популярності. Очевидно, що такі системи зможуть не тільки зменшити вартість онлайн-освіти, а й радикально скоротити час, що витрачається на перевірку завдань.

У цій роботі були розглянуті та програмно реалізовані методи попередньої обробки тексту, конструювання ознак, векторизації тексту, обирання ознак, перехресної перевірки, комбінування базових моделей та класифікації даних. Було проведено дослідження ефективності кінцевої моделі із використанням коефіцієнта “каппа Коена” (англ. Cohen's kappa).

Аннотация

Дипломная работа посвящена решению задачи автоматизированного оценивания произведений небольшого объёма (эссе), с использованием методов и средств компьютерной обработки естественного языка (англ. natural-language processing, NLP). В связи с тем, что в последнее время интерес к самообразованию и массовым открытым онлайн-курсам (англ. massive open online course) постоянно растет, системы, которые позволяют автоматизировать проверку академической успеваемости приобретают большую популярность. Очевидно, что такие системы смогут не только уменьшить стоимость онлайн-образования, но и радикально сократить время, затрачиваемое на проверку задач.

В этой работе были рассмотрены и программно реализованы методы предварительной обработки текста, конструирования признаков, векторизации текста, выбора признаков, перекрестной проверки, комбинирования базовых

моделей и классификации данных. Была проведена оценка эффективности конечной модели с использованием коэффициента «каппа Коэна» (англ. Cohen's kappa).

Abstract

The aim of this thesis is to solve the problem of Automated Essay Scoring, using natural-language processing (NLP). Due to the fact that in recent years the interest in self-education and Massive Open Online Courses (MOOC) is constantly growing, systems that are able to deliver automation in academic progress evaluation are gaining popularity. Obviously, such systems can not only reduce the cost of online education but also radically reduce the time spent on task scoring.

In this work, the methods of text pre-processing, feature engineering, text vectorization, feature selection, cross-validation, ensemble methods, and supervised learning classification were implemented. The performance of the final model was evaluated using Cohen's kappa coefficient.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6427. 02.000 ТЗ	Технічне Завдання	4	
3	А4	ДП 6427. 03.000 ПЗ	Пояснювальна записка	64	
4	А4	ДП 6427. 04.000 Д1	Принципова схема	1	
5	А4	ДП 6427. 05.000 Д2	Функціональна схема	1	
6	А4	ДП 6427. 06.000 Д3	Структурна схема	1	
7	А4	ДП 6427. 07.000 Д4	Лістинг програми	7	

Зм.		№ документа	Підп.	Дата	ДП.6427.01.000 ВП		
Розробив		Фельдман М. Г.			Система автоматичного оцінювання творів Відомість дипломного проєкту	Літ.	Аркуш	
Перевір.		Саверченко В. Г.				Т	1	1
Н.конт		Симоненко В.П.				НТУУ «КПІ імені Ігоря Сікорського», ФІОТ Група ІО - 64		
Затв.								

Технічне завдання

до дипломного проєкту

на тему «Система автоматичного оцінювання творів»

ЗМІСТ

1.НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: «Система автоматичного оцінювання творів».

Область застосування: програма може бути використана у вільному доступі будь-яким користувачем для приблизної оцінки якості англомновного текстовго документу невеликого обсягу.

2.ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3.МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення системи автоматичного оцінювання творів.

4.ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, публікації в спеціалізованих періодичних виданнях, довідники по платформах дистанційного навчання, публікації в мережі Інтернет по даній темі.

5.ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Додаток, що розробляється повинен:

- Бути здатним до перенавчання на іншому наборі даних.
- Перенавчатися не більше однієї години.
- Давати відповідь не більше ніж за п'ять секунд.
- Займати не більше 50-ти мегабайтів дискового простору

5.2. Вимоги до пристрою, де відбувається тренування моделей

- Мінімальний обсяг оперативної пам'яті - 8 ГБ;
- Розмір сховища - 128 Г;
- Частота процесора - 2,4 ГГц;
- Архітектура - 64-бітна.
- Наявність мови програмування Python та бібліотек Anaconda.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення необхідної літератури	19.02.2020
Складання і узгодження технічного завдання	06.03.2020
Написання вступної частини та огляд рішень	19.03.2020
Розробка архітектури додатку	03.04.2020
Написання програмної частини	10.04.2020
Тестування та виправлення помилок	01.05.2020
Оформлення документації дипломного проекту	15.05.2020
Попередній захист та проходження нормативного контролю	29.05.2020
Захист дипломного проекту	15.06.2020

Пояснювальна записка
до дипломного проєкту

на тему: «Система автоматичного оцінювання творів»

Київ – 2020 року

ЗМІСТ

ВСТУП

Розгорнута відповідь у вигляді невеликого письмового твору завжди була й залишається важливою частиною академічного процесу. Особливої важливості така форма відповіді набуває коли йдеться про такі предмети як рідна або іноземна мови та література. В обох випадках на остаточну оцінку впливає як стиль твору (об’єм написаного, різноманіття використаної лексики, граматика, орфографія), так і його змістове наповнення. Очевидно, що у творі з літератури більша увага буде приділена змістовному наповненню, у завданні ж з рідної чи іноземної мови – стилю.

Сьогодні, перевірку розгорнутих письмових відповідей з мови та літератури виконують викладачі, що займає велику частину їх робочого, а подекуди й вільного часу. Звісно, автоматизація цього процесу дуже б їм допомогла. Проте, на жаль, на даному етапі розвитку технологій повної автоматизації цього процесу досягнути не вдається. Проблема полягає в тому, що оцінити саме змістове наповнення написаного є дуже складним завданням, яке на сьогодні (а можливо і назавжди) під силу лише людині. Іншими словами, процес оцінювання змістовного наповнення навіть невеликого тексту, наразі не вдається автоматизувати за допомогою комп'ютерної техніки. В той самий час автоматизація оцінювання стилю написаного (включно навіть із структурою та зв’язністю) є задачею цілком реальною та дуже актуальною. Саме тому, коли мова йде про системи автоматизовного оцінювання текстових документів, здебільшого мають на увазі оцінювання їхньої стилістичної складової. Такі системи прийнято називати системами автоматичного оцінювання творів (англ. automated essay scoring (AES) systems).

Автоматизоване оцінювання есе (AOE) - це використання спеціалізованих комп’ютерних програм для присвоєння оцінок текстовим документам, написаним у навчальних умовах. Метою є класифікація великого набору текстових сутностей на невелику кількість дискретних категорій, що

відповідає можливим оцінкам (класам), наприклад, числам від 1 до 5. Отже, це можна вважати проблемою статистичної класифікації.

Актуальність теми даної роботи обумовлена тим, що останнім часом набувають великої популярності різноманітні сервіси для самоосвіти, відкриті онлайн-курси та національні екзамени на кшталт українського зовнішнього незалежного оцінювання. В усіх трьох випадках одним із можливих завдань є саме невеликий прозовий твір. Через масовість таких методик навчання та перевірки знань (наприклад ЗНО одночасно складають сотні тисяч школярів), їх організатори стикаються з багатьма проблемами, основними з яких є кількість викладачів, необхідних для перевірки творів, та час, необхідний для цього. Автоматизація процесу перевірки (навіть часткова) дозволить, як зменшити кількість викладачів, так і скоротити час. Зрозуміло, що зменшення кількості перевіряючих має наслідком зниження вартості вартості заходу для його організаторів, а скорочення часу, необхідного для перевірки, робить методи масового навчання та оцінювання знань більш популярними серед учнів.

Метою даної роботи є аналіз та імплементація засобів, методів та моделей, які можуть бути використані для створення системи автоматичного оцінювання невеликих прозових творів. Система яка досліджується та розробляється в даній роботі повинна мати можливість автоматично оцінити стилістичну якість текстового документа, який не був присутній у навчальній вибірці (даних, які були надані системі для її налаштування під час розробки). Відповідь системи (оцінка) повинна належати до множини дискретних категорій, наприклад, до множини чисел від 1 до 5 (множина оцінок залежить від навчальної вибірки). Ступінь згоди між оцінками, даними системою та оцінками викладачів повинна бути істотною.

РОЗДІЛ 1
ОСНОВИ ТЕОРІЇ КЛАСИФІКАЦІЇ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Як вже було зазначено у Вступі, можна розглянути процес оцінювання текстових документів з точки зору розподілення великого набору текстових сутностей на невелику кількість дискретних категорій, що відповідають можливим оцінкам. Іншими словами, цей процес можна вважати проблемою статистичної класифікації.

У статистиці класифікація - це проблема визначення того, до якої з категорій (підгруп) належить нове спостереження (у нашому випадку твір) на основі навчального набору даних (навчальної вибірки), що містить спостереження (або випадки), належність до категорії яких відома. Алгоритми, що вирішують завдання статистичної класифікації називають – класифікаторми. Такі алгоритми є дуже популярними, відомими прикладами є спам-фільтри та класифікатори зображень. У цьому розділі будуть розглядатися існуючі засоби, етапи та методи, імплементація яких допоможе у побудові класифікатора невеликих прозових творів.

1.1 Машинне навчання

Машинне навчання - це підрозділ інформатики, який стосується побудови алгоритмів які, для своєї роботи, спираються на збірку прикладів якогось явища. Ці приклади можуть походити з природи, бути створеними людиною або породжуватися за допомогою іншого алгоритму. Машинне навчання також можна визначити як процес вирішення практичної задачі, яка потребує як мінімум двох етапів: створення набору даних та алгоритмічної побудови статистичної моделі на основі цього набору.

Необхідно також пояснити, що значить “навчання” у даному контексті. Класичне визначення надається у [1]: “Кажуть, що комп'ютерна програма

навчається досвіду E щодо деякого класу завдань T та показнику ефективності P, якщо її ефективність у завданнях T, виміряна P, покращується з досвідом E.” У різних випадках T, P і E можуть посилатися на дуже різні речі. Одними з найпопулярніших завдань T у машинному навчанні можна назвати:

- класифікацію екземпляра до однієї з категорій за ознаками;
- регресію – прогнозування числової цільової ознаки на основі інших особливостей екземпляра;
- кластеризацію – групування екземплярів на основі особливостей цих екземплярів, щоб члени всередині груп були більш схожими один на одного, ніж на екземпляри в інших групах;
- виявлення аномалії – пошук екземплярів, які "сильно відрізняються" від решти вибірки або від групи екземплярів в якій вони знаходяться;

Як приклад використання методів машинного навчання часто наводиться алгоритм спам-фільтру. Спам-фільтр - це програма, яка може навчитися виявляти спам-листи, якщо їй будуть наведені приклади таких листів (наприклад, листи позначені користувачами як спам) та приклади звичайних листів. Сукупність прикладів, які система використала для навчання - називається навчальним набором. Кожен такий приклад називається навчальним екземпляром (або зразком). У цьому випадку завдання T - позначити спам для нових електронних листів, досвід E - це усі приклади спам та не-спам листів, також потрібно визначити міру ефективності P; наприклад, ви можете використовувати частку правильно класифікованих електронних листів. Цей конкретний показник ефективності називається точність і часто використовується в задачах класифікації.

Навіщо використовувати саме машинне навчання? Щоб відповісти на це питання, використаємо приклади спам-фільтрів описані у [2]. Розглянемо реалізацію, яка використовує традиційні методи програмування (рис. 1.1):

1. Подивимося, як зазвичай виглядає спам. Можемо помітити деякі слова та фрази (наприклад, "кредитна картка", "безкоштовно" та

"дивовижно"), які зустрічаються доволі часто. Можливо, ви також помітили б кілька інших закономірностей в імені відправника, тілі електронної пошти, тощо.

2. Напишемо алгоритм виявлення для кожного з шаблонів, які ми помітили, і наша програма позначить електронні листи як спам, якщо ряд цих шаблонів виявлено.
3. Ми будемо постійно тестувати нашу програму поки не досягнемо бажаної точності виявлення.

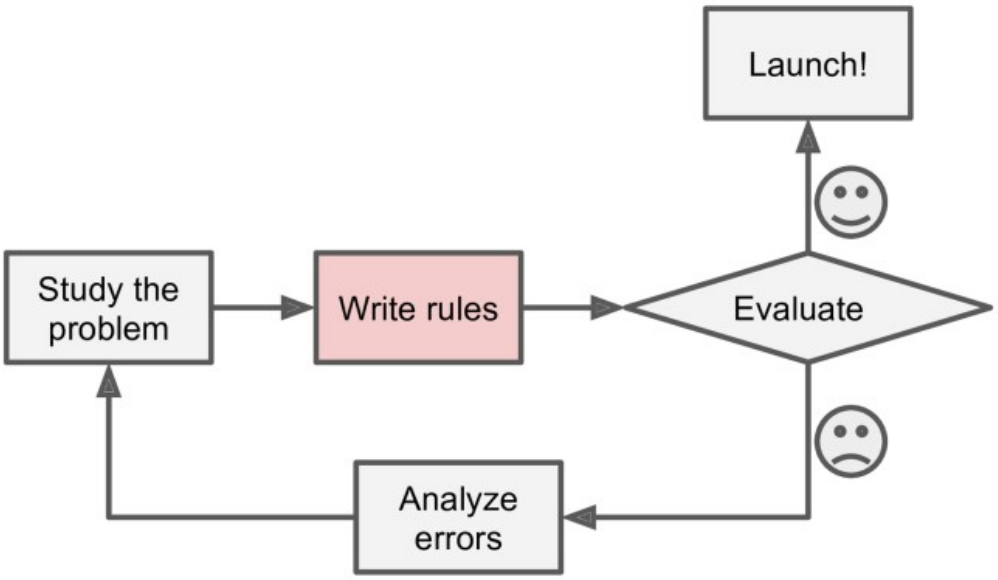


Рис. 1.1 – Традиційні методи програмування[2].

Оскільки проблема не тривіальна, програма, ймовірно, буде довгим переліком складних правил, що досить важко підтримувати. На відміну від цього, фільтр спаму, заснований на техніці машинного навчання, автоматично навчається які слова та фрази є хорошими прогнозами спаму, виявляючи незвично часто використовувані слова та фрази у прикладах спам-листів (рис. 1.2). Така програма набагато коротша, простіша в обслуговуванні і, швидше за все, навіть більше точна.

Більш того, якщо зловмисники помітять, що наприклад при використанні слова “безкоштовно” їх листи потрапляють до спаму, вони просто почнуть писати “безоплатно”, обійшовши таким чином одне з правил нашого традиційного алгоритму. Таким чином зловмисники зможуть обійти

усі наші правила. Звісно, можна написати нові правила (вже з урахуванням нових слів), зловмисники ж почнуть обходити й їх. Отже, потрібно буде постійно працювати над все новими й новими правилами, щоб традиційна програма працювала. Якщо ж використовувати методи машинного навчання для вирішення цієї задачі, програма автоматично помітить, що тепер слово “безоплатно” почало зустрічатися дуже часто у листах, помічених користувачами як спам (рис. 1.3.).

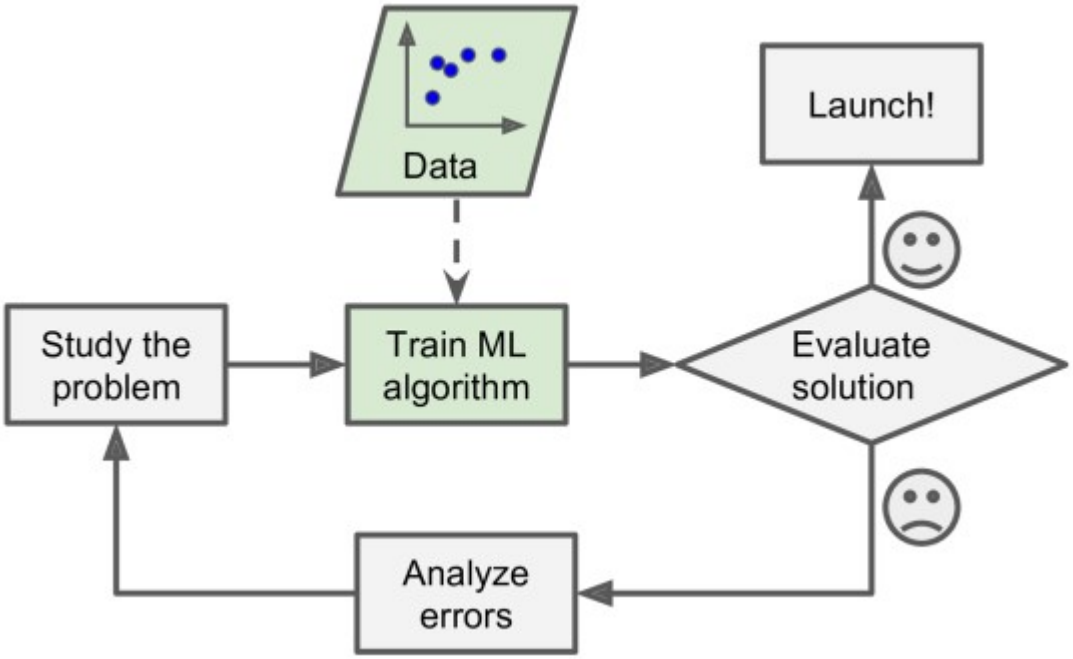


Рис. 1.2 – Метод за використанням техніки машинного навчання[2].

Звісно, використання алгоритмів машинного навчання також є виправданим, коли проблема взагалі не піддається вирішенню з використанням класичних алгоритмів. Класичними прикладами таких задач є розпізнавання людської мови та різноманітні проблеми, що пов’язані з використанням комп’ютерного зору.

Серед великої кількості алгоритмів машинного навчання, можна виділити ті, що для своєї роботи потребують наявності категорії до якої належить кожен з прикладів навчального набору. Такий набір даних називають розміченим. Наприклад, кожному листу з навчального набору, який ми надали нашому алгоритму спам-фільтра, буде відповідати одна з двох категорій: “спам” або “не спам”. Навчання, яке потребує такого

(розміченого) набору даних називають – навчанням з учителем (англ. supervised learning), або контрольованим навчанням. Далі у цій роботі піде мова, здебільшого, саме про цей клас машинного навчання.

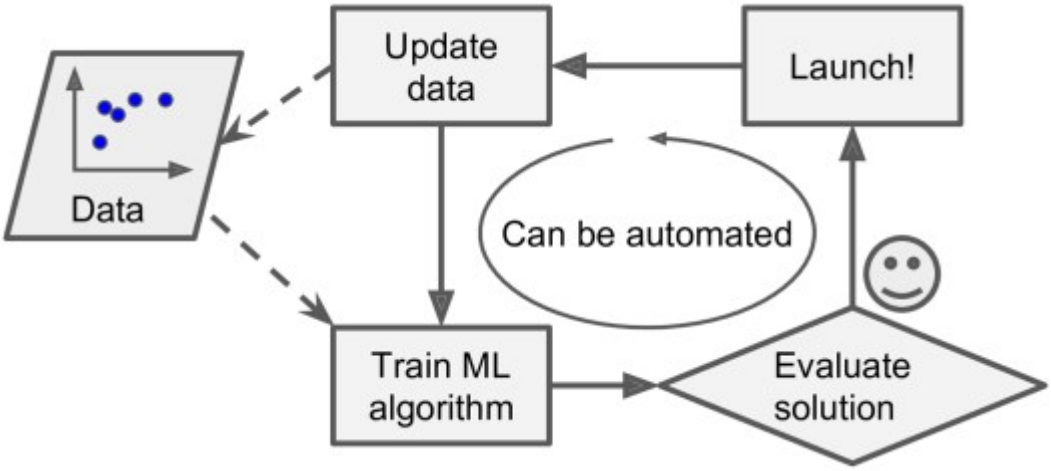


Рис. 1.3 – Автоматична адаптація до змін[2].

1.2 Навчання з учителем

Як можна побачити у [3], у навчанні з учителем навчальній набір – це колекція розмічених прикладів$[(x_i, y_i)]_{(i=1)}^{N}$. Кожен x_j з N є вектором ознак (англ. feature vector). Вектор ознак – це такий вектор у якому кожен його вимір $j=1,...,D$ містить значення, що якось описує приклад. Це значення називають ознакою (англ. feature) та позначаютьx^j. Наприклад, якщо кожен приклад x у нашій колекції представляє собою людину, то перша ознака, x^1, може містити її висоту в см, друга, x^2 , може містити вагу в кг, x^3, може містити стать, тощо. Для усіх прикладів у наборі, *j-та* ознака з вектору ознак завжди містить однаковий вид інформації (наприклад вагу в кг). Мітка (англ. label) y_i може належати, як до множини дискретних категорій (класів) {1, 2, . . . , C}, так і до множини реальних чисел. У першому випадку мова йде про алгоритми класифікації (рис. 1.4), у другому – регресії (рис. 1.5). Мета алгоритму навчання з учителем полягає у використанні навчального набору даних для створення моделі, яка згодом зможе приймати вектор ознак *x,* якого не було у цьому наборі на вхід, а на виході видавати мітки для цього вектора ознак. Наприклад, модель, створена за допомогою навчального набору даних

результатів аналізів людей, могла б взяти як вхід вектор ознак їх аналізів, а на виході дати ймовірність того, що людина має певну хворобу.

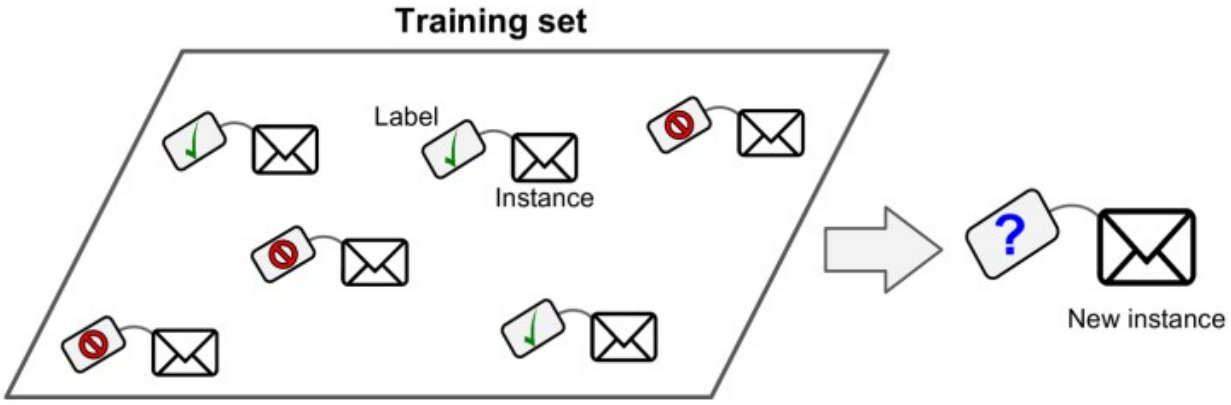


Рис. 1.4 – Приклад розміченого навчального набору для задачі класифікації[2].

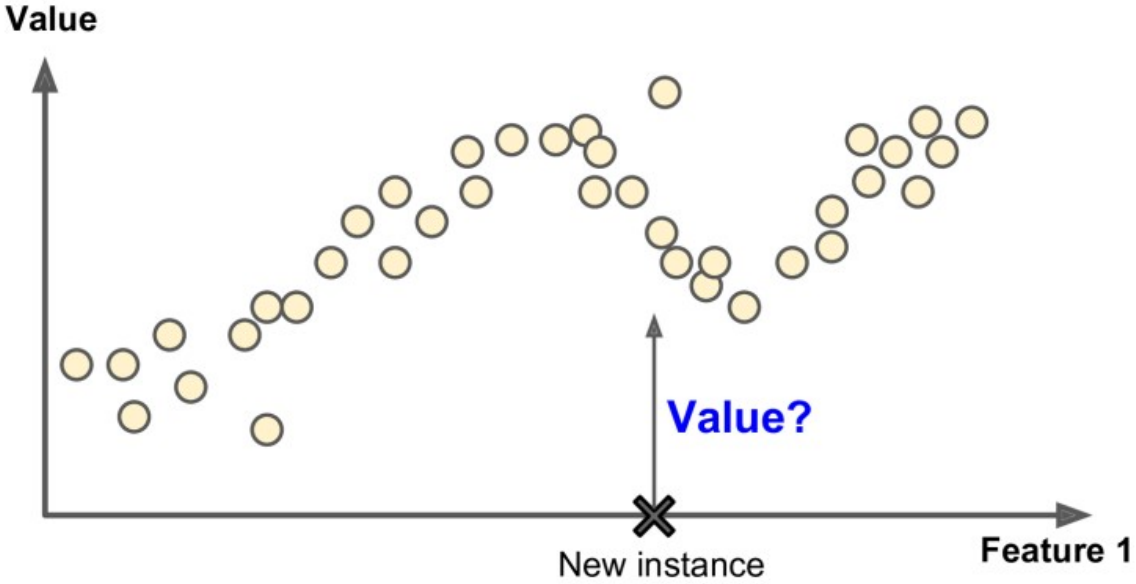


Рис. 1.5 – Приклад задачі регресії[2].

Популярні алгоритми, що використовують навчання з учителем:

- *k*-найближчих сусідів (англ. k-nearest neighbor method)
- Лінійна та логістична регресії
- Метод опорних векторів
- Дерева рішень
- Випадковий ліс (англ. random forest)
- Дуже (екстремально) випадковий ліс (англ. ExtraTrees)
- Адаптивний та градієнтний бустінг (англ. adaptive and gradient boosting)

- Нейронні мережі

Оскільки у даній роботі буде використана велика кількість саме алгоритмів навчання з учителем, необхідно коротко пояснити загальні принципи їх роботи. Як приклад, можна знову використати проблему створення спам-фільтру описану в [2].

Процес контрольованого навчання починається зі збору даних. Як вже було зазначено вище дані для такого виду навчання є сукупністю пар (вхід, вихід). Входи можуть бути будь-чим, наприклад, електронною поштою, зображеннями або вимірами якогось датчика. Виходами, як правило, є реальні числа або мітки (англ. labels). Отже, був знайдений потрібний нам набір даних, наприклад, десять тисяч листів електронної пошти, кожний лист має мітку “spam” або “not_spam” (якщо міток не було одразу, ми могли б розмітити дані самостійно або знайти компанію, що спеціалізується на цьому). Тепер потрібно перетворити кожен наш лист (кожен вхід) у вектор ознак. Існує багато методів перетворення текстової інформації у такий вектор, одним із них є метод на основі “торби слів” (анг. bag of words). Принцип роботи дуже простий. Створюється банк з усіх слів, що є в листах (торба слів), наприклад, вийшло 20 тисяч слів. Тоді кожному листу в датасеті буде відповідати вектор у 20 тисяч ознак. Кожна ознака конкретного листа буде рівна 0 або 1, в залежності від того чи є слово, що відповідає цій ознаці (наприклад, слово “money”) у цьому листі. Зрозуміло, цю операцію повторюють для кожного листа в колекції, що дає 10 тисяч векторів ознак (кожний вектор довжиною у 20 тисяч ознак). Якщо цього вимагає конкретний алгоритм, також можна замінити мітки на відповідні натуральні числа, наприклад, “spam” = 1, “not_spam” = -1. Тепер дані готові до машинного навчання.

У даному прикладі у якості класифікатора буде використаний **метод опорних векторів**. Метод бачить кожен вектор ознак як точку у високомірному просторі (у нашому випадку простір - 20000-мірний). Алгоритм накладає всі вектори ознак на уявний 20000-розмірну площину і

малює уявну 20 000-мірну лінію (гіперплощину), яка відокремлює приклади з позитивними мітками від прикладів з негативними мітками. У машинному навчанні межа, що розділяє приклади різних класів, називається межею рішення. Рівняння гіперплощини задається двома параметрами, реальним значенням вектора *w* такої ж розмірності, як і наш вхідний вектор ознак *x*, і реальним числом b, та виглядає як: *wx - b=0,* де вираз *wx* означає $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \cdots, + w^{(D)}x^{(D)}$, та *D* є розмірністю вектора ознак. Тепер передбачувана мітка для деякого вектора ознак x задається так: *y = sign(wx-b),* де *sign* є математичним оператором який приймає будь-яке значення на вхід та повертає +1, якщо значення було позитивним, або -1, якщо воно було негативним. Мета алгоритму навчання в даному випадку полягає у використанні набору даних та пошуку оптимальних значень *w** і *b** для параметрів *w* і *b*. Як тільки алгоритм навчання виявить ці оптимальні значення модель *f(x)* визначається як: *f(x)=sign(w*x-b*).* Отже, щоб передбачити, чи є повідомлення електронної пошти спамом за допомогою метод опорних векторів, вам потрібно взяти текст повідомлення, перетворити його у вектор ознак, а потім помножити цей вектор на *w** та відняти *b** .Знак результату дасть нам прогноз (+1) означає "спам", - 1 означає "not_spam")[3].

Проте, як машина знаходить *w** і *b** ? Вона вирішує оптимізаційну задачу. Машини добре оптимізують функції при обмеженнях. Які ж обмеження задовольняють нас у цьому випадку? Ми пам’ятаємо, що маємо правдиві мітки для усіх листів, адже наші дані це колекція розмічених прикладів $\{(x_i, y_i)\}_{i=1}^{N}$. Тоді обмеження стають очевидними:

- $wx_i - b > 1 if y_i = +1$, та
- $wx_i - b < -1 if y_i = -1$.

Хотілося б, щоб розділення (англ. margin) було якомога кращим. Можна оцінити міру розділення дивлячись на відстань між найближчими прикладам з двох класів, адже поділ на ці класи був здійснений завдяки нашій розділовій гіперплощині. Чим кращого розділення ми доб’ємося, тим краще потім модель буде класифікувати нові приклади. Інакше кажучи, ми покращуємо

узагальнення моделі. Щоб досягнути цього потрібно мінімізувати Евклідову норму w, що визначається як:

$$\|w\| = \sqrt{\sum_{j=1}^{D} (w^{(j)})^2}$$

Отже, остаточна проблема оптимізації, яку потрібно вирішити буде виглядати так : мінімізувати$\|w\|$заумови$y_i(wx_i - b) \geq 1 for i = 1,..,N$. Рішення цієї оптимізаційної проблеми (отримання w^* і b^*) називають статистичною моделлю (або просто моделлю). Процес побудови такої моделі називають тренуванням. Якщо вектори ознак мають лише два виміри, можна легко візуалізувати нашу проблему (рис. 1.6).

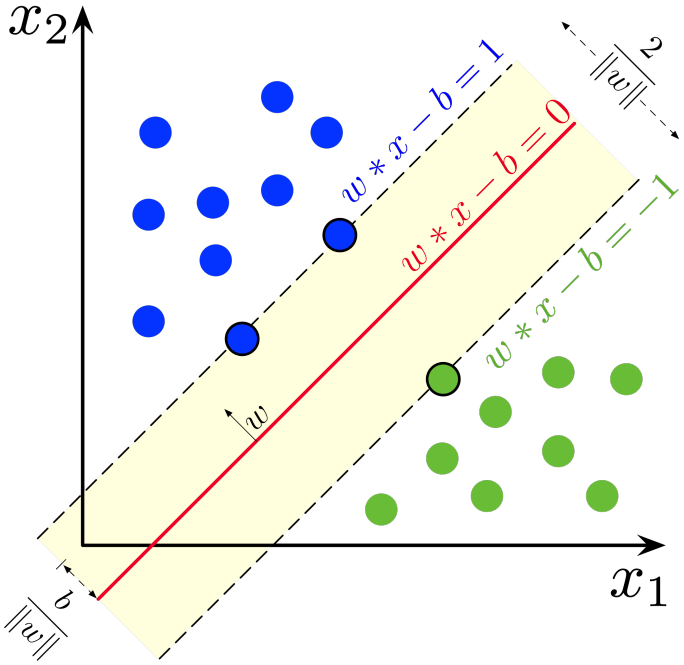


Рис 1.6 – Приклад гіперплощини для двовимірних векторів ознак.

Чому модель здатна правильно передбачити мітки нових, раніше небачених прикладів? Щоб зрозуміти це, треба подивіться на рис. 1. 6 . Якщо два класи відокремлені один від одного межею рішення, то, очевидно, приклади, що належать до кожного класу, розташовані у двох різних підпросторах, які створює ця межа. Якщо приклади, що використовуються для навчання, вибиралися випадковим чином, незалежно один від одного і дотримуючись тієї ж процедури, то, статистично, більш ймовірно, що новий негативний приклад буде розміщений на ділянці десь недалеко від інших

негативних прикладів. Те саме стосується нового позитивного прикладу: він, ймовірно, буде поряд інших позитивних прикладів. У такому випадку межа рішення, з великою ймовірністю, добре відокремлюйте один від одного нові позитивні та негативні приклади. Для інших, менш вірогідних ситуацій наша модель буде робити помилки, але оскільки такі ситуації є менш імовірними, кількість помилок, ймовірно, буде меншою, ніж кількість правильних прогнозів.

Інтуїтивно зрозуміло, що чим більший набір навчальних прикладів, тим більше малоймовірно, що нові приклади будуть сильно відрізнятися і лежати на площині далеко від прикладів, що використовуються для тренувань (навчання). Щоб мінімізувати ймовірність помилок на нових прикладах, алгоритм методу опорних векторів, шукаючи найкращого розділення явно намагається провести межу рішення таким чином, щоб вона лежала якнайдалі від прикладів обох класів.

Така реалізація методу опорних векторів є дуже наївною та не бере до уваги багато важливих нюансів, наприклад побудову нелінійних гіперплощин або регуляризацію. Однак, такий опис методу дає змогу зрозуміти основні принципи навчання з учителем: будь-який алгоритм класифікації, що будує модель неявно або явно створює межу рішення. Межа рішення може бути прямою, або вигнутою, або вона може мати складну форму, або це може бути суперпозиція деяких геометричних фігур. Форма межі рішення визначає точність (це співвідношення прикладів, мітки яких були прогнозовані правильно) або іншу метрику якості моделі. Форма межі рішення та спосіб алгоритмічного чи математичного обчислення на основі навчальних даних відрізняє один алгоритм машинного навчання від іншого.

На практиці існує два інших важливих параметра алгоритмів навчання: швидкість побудови моделі та час обробки прогнозу. У багатьох практичних випадках ви б віддали перевагу алгоритму навчання, який швидко будує менш точну модель. Крім того, ви можете віддати перевагу менш точній моделі, яка набагато швидше робить прогнози.

1.3 Навчання без учителя

У навчанні без учителя набір даних - це сукупність нерозмічених прикладів $\{(x_i)\}_{i=1}^{N}$. Знову ж таки, x - вектор ознак, а мета непідтримуваного алгоритму навчання – це створити модель, яка приймає векторний елемент *x* як вхідний і перетворює його в інший вектор або у значення, яке може бути використане для вирішення практичної проблеми. Наприклад, при кластеризації (рис. 1.7) модель повертає ідентифікатор кластера для кожного вектора ознак у наборі даних. При зменшенні розмірності та візуалізації вихідна модель - це вектор ознак, який має меншу розмірність, ніж оригінал; у виявленні викидів (англ. outliers) вихідний сигнал - це реальне число, яке вказує, наскільки *x* відрізняється від "типового" прикладу в наборі даних[3].Популярними представниками алгоритмів навчання без учителя є:

- Кластеризація
  - метод k-середніх
- Візуалізація та зменшення розмірності
  - метод головних компоне́нт
    - t-розподілене вкладення стохастичної близькості

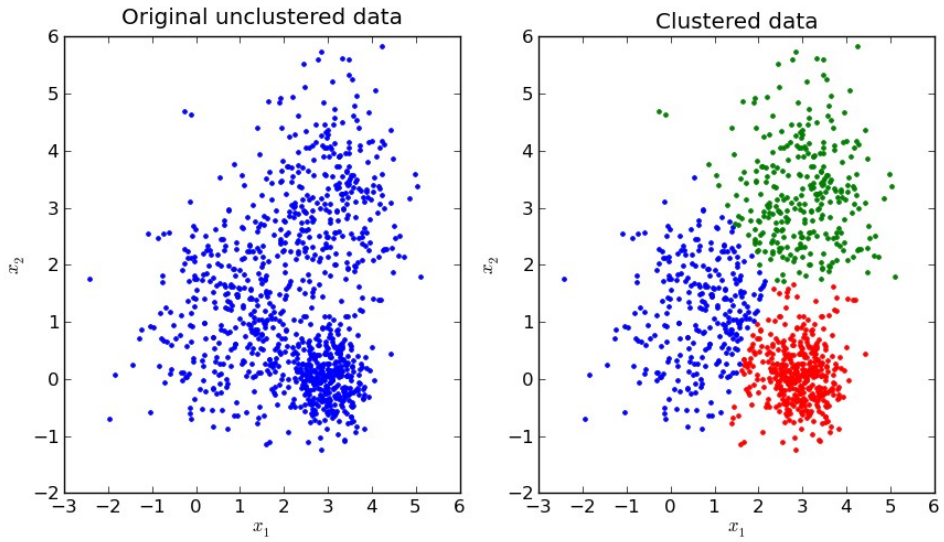


Рис 1.7 – Приклад кластеризації.

Висновки до розділу 1

В даному розділі було коротко висвітлено основи теорії класифікації та регресії з використання машинного навчання (як з учителем так і без нього). Було обґрунтовано використання саме методів машинного навчання для вирішення завдань пов’язаних з текстовою класифікацією.

РОЗДІЛ 2
ОПИС МОДЕЛЕЙ, МЕТОДІВ ТА ЕТАПІВ

2.1 Попередня обробка тексту

Оскільки в якості входів у системі будуть виступати текстові документи, потрібно розібрати такий важливий процес, як попередня обробка текстових даних (англ. text preprocessing). Найбільш простими етапами попередньої оброки можна назвати приведення всіх слів до нижнього регістру та видалення всіх знаків пунктуації. Далі будуть розглянуті більш складні етапи.

Токенізація (анг. Tokenization) [4] – це поділ тексту на окремі змістовні частини, наприклад, слова або речення. Інакше кажучи, це просто сегментація тексту. У цій роботі (та й взагалі найчастіше) мова буде іти саме про поділ текстового документу на список слів (рис 2.1). Це завдання, спершу, може здатися дуже простим, однак, якщо ви спробуєте реалізувати алгоритм самостійно, то швидко зрозумієте, що ця проблема не є тривіальною. Наприклад, крапка, як роздільник для токенізації на речення вийде з ладу, якщо у вас є абревіатури з крапками. Тож вам доведеться мати більш складну модель, щоб досягти достатньо хорошого результату. Зазвичай ця проблема вирішується за допомогою бібліотек nltk або spacy.

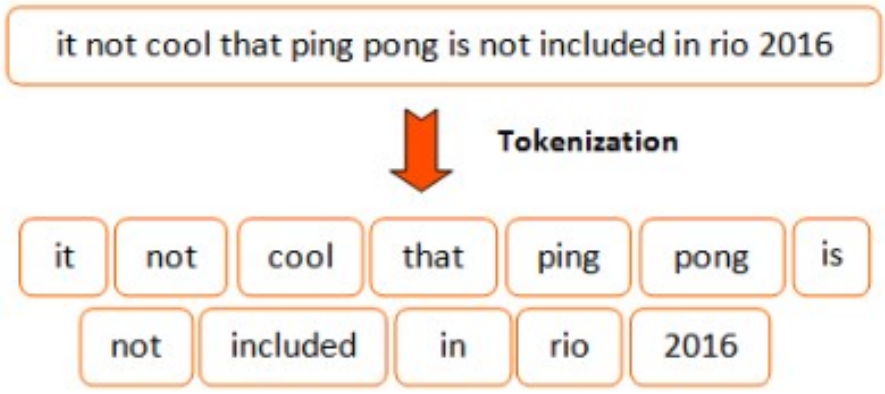


Рис. 2.1 – Приклад токенізації на слова.

Видалення стоп-слів (англ. stop words removal) [5] – це процес видалення слів, що дуже часто зустрічаються в мові на якій написані документи, або просто є часто використовуваними в конкретному навчальному наборі. Наприклад, слово “або” є просто дуже вживаним в українській мові, слово ж “змінна” дуже часто використовується у текстах стосовно програмування. Оскільки, що вважати за стоп-слово сильно залежить від конкретної задачі не існує єдиного списку таких слів. На практиці використовують списки з бібліотек nltk та spacy, а вже до них додають нові слова, що розробники вважають часто вживаними у конкретній задачі.

Стемінг (англ. steaming) [6] – це процес грубого скорочення слова до його основи. “Грубість” методу виражається у тому, що отримана основа не обов’язково має співпадати з морфологічний коренем заданого слова. Цей метод доволі погано працює з мовами, що мають складну морфологію, адже звичайне відсічення закінчень утворює основи, що не мають сенсу або мають інше значення. Прекрасним прикладом такої мови є українська, де слово “вчаться ” може бути скорочено до “вча”.

Лематизація (англ. lemmatization) [6] – це більш гнучкий спосіб скорочення слова до його основи. Він використовує словник та морфологічний аналіз, щоб знайти правильну форму – лему. Цей спосіб є більш складним в реалізації, ніж стемінг але дає більш точні результати. Процес лематизації є дуже складним та потребую глибокого знання морфології мови. Зазвичай використовуються готові рішення на основі бібліотек nltk або spacy.

Розмічування частин мови (англ. part-of-speech tagging) [7] – це процес віднесення слова в текстовому документі належним до певної частини мови, заснований як на його значенні, так і на контексті в якому воно перебуває – тобто, на його зв'язку з суміжними та спорідненими словами у фразі, реченні, або абзаці (рис. 2.2). Як і у випадку з лематизацією, процес потребує глибинного знання морфології мови.

Розпізнавання іменованих сутностей (англ. named-entity recognition) [8] – це ціла окрема область у завданні видобування інформації з текстових документів, що намагається знайти і класифікувати іменовані сутності, такі як імена людей або організацій, місця, географічні назви, тощо.

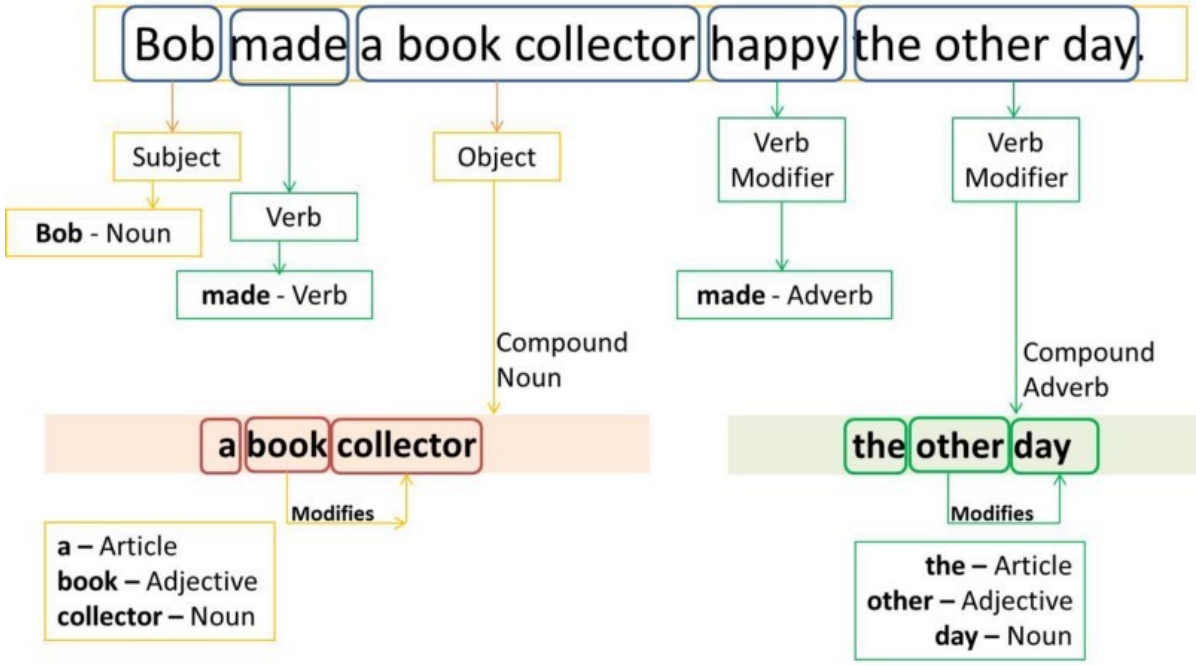


Рис. 2.2 – Приклад розмічування частин мови.

2.2 Конструювання ознак

На практиці дані дуже рідко знаходяться у вигляди готових векторів ознак, та навіть якщо це так, завжди є можливість покращити результати моделі, отримавши нові ознаки з вже існуючих. Цей процес прийнято називати – процес конструювання ознак (англ. Feature Engineering). Для вдалого створення нових ознак необхідно мати глибокі знання з предметної галузі. Конструювання ознак є важливим етапом машинного навчання, і є як складним та витратним. У часи, коли майже всі алгоритми вже реалізовані у вигляди готових до використання бібліотек, саме процес вдалого створення та комбінування ознак виділяє хорошого спеціаліста з машинного навчання. Якщо ж говорити більш конкретно про проблему, що стоїть перед нами у цій роботі, то перед нами також стоїть **завдання векторизації** текстової інформації. Найбільш популярними алгоритмами для вирішення цього

завдання є модель «торба слів» (англ. bag-of-words) та TF-IDF (від англ. TF – term frequency, IDF – inverse document frequency) модель. Є також багато інших способів (в основному із застосуванням нейронних мереж), які на жаль не будуть розглядатися у цій роботі (наприклад, word2vec модель).

Модель «торба слів» [9] – це один з найпопулярніших способів перетворити текст на вектор ознак. Принцип роботи моделі вже був описаний у цьому проекті (приклад реалізації спам-фільтру з використанням методу опорних векторів). Дуже стисло, створюється вектор із довжиною словника (всі слова з усіх прикладів), обчислюємо кількість входів кожного слова в прикладі та розміщуємо цю кількість входів у відповідному положенні у векторі. Перед тим як подати текстові дані до цієї моделі, з ними обов'язково потрібно провести процес попередньої обробки (рис. 2.3).

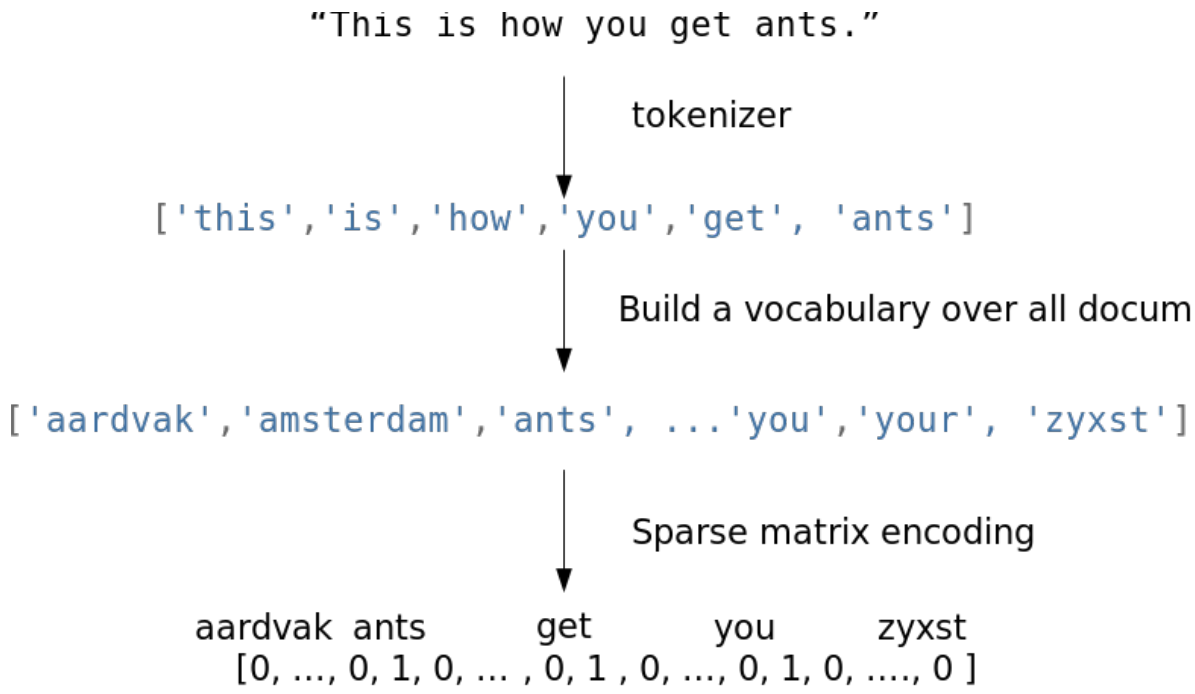


Рис. 2.3 – Приклад реалізації найпростішої моделі "торби слів".

Така реалізація є дуже примітивною оскільки вона не бере до уваги порядок слів у тексті, наприклад, фрази "i have no cows" (в мене немає коров) та "no, i have cows" (в мене є корови) будуть векторизовані однаково. Щоб уникнути цього під час токенізації використовують, так-звані, n-грами (послідовність n токенів). Також проблемою є те, що всі слова мають однакову вагу, що не є логічним, оскільки слова, що часто вживаються в усіх прикладах менше

впливають на мітку ніж унікальні слова. Цю проблему вирішує вдосконалена версія моделі – TF-IDF.

TF-IDF (term frequency-inverse document frequency), як зазначено у [10], працює так: слова, які рідко зустрічаються в корпусі (у всіх документах цього набору даних), але присутні в цьому конкретному документі, можуть бути важливішими. Тоді є сенс збільшити вагу більшої кількості доменних слів, щоб відокремити їх від загальних слів. Формалізуємо написане. TF (term frequency – частота слова)[10] – відношення числа входжень обраного слова до загальної кількості слів документа. Таким чином, оцінюється важливість слова t_i в межах обраного документа. $TF = \frac{n_i}{\sum_k n_k}$, де n_i є число входжень слова в документ, а в знаменнику – загальна кількість слів в документі. IDF (inverse document frequency – обернена частота документа) – інверсія частоти, з якою слово зустрічається в документах колекції. Використання IDF зменшує вагу широковживаних слів. $IDF = \frac{\log|D|}{|d_i \supset t_i|}$, де |D| – кількість документів колекції; $|d_i \supset t_i|$– кількість документів, в яких зустрічається слово t_i, коли $n_i \neq 0$. Вибір основи логарифму у формулі не має значення, адже зміна основи призведе до зміни ваги кожного слова на постійний множник, тобто вагове співвідношення залишиться незмінним. Іншими словами, показник TF-IDF це добуток двох множників: TF та IDF. Більшу вагу TF-IDF отримають слова з високою частотою появи в межах документа та низькою частотою вживання в інших документах колекції. З'являються також і нові підходи. Найпопулярнішим методом нової хвилі є Word2Vec, але є також кілька альтернатив (GloVe, Fasttext тощо).

Word2Vec [11] – це особливий випадок алгоритмів векторного представлення слова. Використовуючи Word2Vec та подібні моделі, можна не лише векторизувати слова у просторі (як правило, кілька сотень вимірів), а й порівняти їх семантичну схожість. Це класичний приклад операцій, які можна виконати на векторизованих поняттях: король - чоловік + жінка = королева. Варто зазначити, що ця модель не розуміє значення слів, а просто

намагається розташувати вектори так, щоб слова, які вживаються у загальному контексті, були близькі один до одного. Також треба пам'ятати, що такі моделі потрібно навчати на дуже великих наборах даних, щоб векторні координати фіксували семантику. Це частково вирішується завантаженням пре-тренованих моделей.

2.3 Нормалізація ознак

Нормалізація та зміна розподілу векторів ознак навчального набору є важливим етапом у процесі підготовки даних до подання в модель машинного навчання. Монотонний вигляд векторів ознак є критичним для деяких алгоритмів і не впливає на інші. Це одна з причин підвищеної популярності дерев рішень та всіх похідних алгоритмів (випадковий ліс, градієнтний бустінг). Не завжди є час та бажання розбиратися з перетвореннями, і саме ці алгоритми є найбільш стійкими до незвичних розподілів. Параметричні методи (наприклад, лінійна та логістична регресії, метод опорних векторів) зазвичай вимагають мінімум симетричного та унімодального розподілу даних, що не завжди наведено в реальних даних. Ще одним відомим прикладом алгоритму, що дуже залежить від розподілу ознак є k-найближчих сусідів. Модель буде прогнозувати повну нісенітницю, якщо ознаки будуть розподілені дуже по-різному, тому-що буде майже неможливо виміряти та порівняти відстані між прикладами, що є основою цього алгоритму. Наприклад, завдання полягає в тому, щоб передбачити вартість квартири, використовуючи дві ознаки - відстань від центру міста та кількість кімнат. Кількість кімнат рідко перевищує 5, тоді як відстань від центру міста легко може бути виміряна тисячами метрів.

Найпростішим та найпопулярнішим методом нормалізації є використання **стандартизованої оцінки** (z-оцінки, англ. standard score, z-score). Реалізація відбувається за формулою $\frac{x-\mu}{\sigma}$, де $(x-\mu)$є різницею між

вектором ознаки та его математичним очікуванням, а $\frac{x}{\sigma}$ – це відношення значень цього вектора до его стандартного відхилення (рис 2.4).

У практичних завданнях, будь-яка множина даних X із середнім значенням M і стандартним відхиленням S може бути перетворена в іншу множину із середнім 0 і стандартним відхиленням 1 таким чином, що перетворені значення Z будуть безпосередньо виражатися в відхиленнях вихідних значень від середнього, виміряних в одиницях стандартного відхилення .

Належність z-оцінок до стандартного нормального розподілу забезпечує можливість застосування z-оцінок для порівняння неоднорідних значень первинних вимірювань. Більшість статистичних методів ґрунтуються на припущенні про нормальність розподілу даних, тому застосування z-оцінок спільно з трансформацією до нормального розподілу значно розширює можливості для практичного аналізу і досліджень.

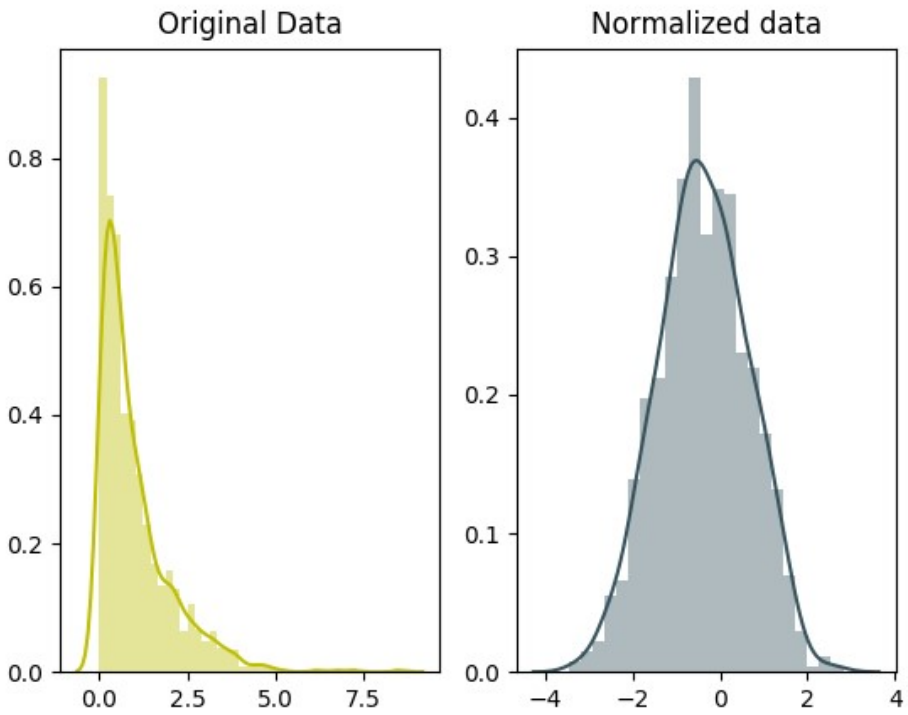


Рис. 2.4 – Приклад нормалізованого вектору ознаки

Також популярним є метод **мин-макс нормалізації** (англ. min-max normalization). Це один з найпростіших методів він полягає у зміні діапазону

значень векторів ознак та зведенні його до [0, 1] або [−1, 1]. Вибір цільового діапазону залежить від характеру даних. Загальна формула для min-max [0, 1] подається у вигляді: $x' = \frac{x - min(x)}{max(x) - min(x)}$. Наприклад, припустимо, що у нас є дані про вагу студентів. Їх вага лежить в діапазоні - [40 , 100] кілограмів. Щоб змінити масштаб цих даних, ми спочатку віднімаємо 40 від ваги кожного учня і поділимо результат на 60 (різниця між максимальною і мінімальною вагою), очевидно, що наш діапазон прийме вигляд [0 , 1]. Можна звести діапазон до будь-якого вигляду за формулою : $x' = \frac{(x - min(x)) - (a - b)}{max(x) - min(x)}$, де a та b є бажаним мінімальним та максимальним значення діапазону відповідно.

Методи стандартизованої оцінки та мин-макс нормалізації мають схожі алгоритми та часто є взаємозамінними. Однак, якщо алгоритм передбачає обчислення відстаней між точками або векторами, вибір за замовчуванням - стандартизована оцінка. Нормалізація мин-макс, в свою чергу, дуже часто використовують для потреб візуалізації, адже вона може звести діапазон розподілу даних до будь якого інтервалу.

Як вже зазначалося раніше створення ознак є важливою частиною аналізу даних. Звісно, у цьому завданні потребується знання кожної конкретної предметної області, однак, все-ж таки існують деякі загальновживані методи. Наприклад, візьмемо проблему створення системи рекомендації нерухомості. Скажімо в нашому навчальному наборі є такі ознаки: кількість кімнат у квартирі та її ціна. Евристично, можна здогадатися, що ознака “ціна за кімнату” може позитивно вплинути на якість роботи моделі. Такі прийоми називають “взаємодією ознак”. Під такою взаємодією може стояти будь-яка арифметична (або взагалі математична) операція. Слід пам’ятати, що утворена таким способом ознака зовсім не обов’язково повинна мати фізичний зміст, щоб покращити результати моделі. Беручи це до уваги, зрозуміло, що при малій кількості початкових ознак, гарною тактикою є генерація всіх можливих взаємодій, а потім відбір тих, що

покращують результати роботи моделі. Процес відбору “корисних” ознак і буде описаний у наступному підрозділі.

2.4 Відбір ознак

Чому взагалі потрібно відбирати ознаки? Спочатку ця ідея може здатися протиінтуїтивною, адже, чим більше різноманітних ознак, тим краще модель буде навчатися. Проте є як мінімум дві важливі причини, щоб позбутися неважливих особливостей буває вкрай необхідно. Перша – зрозуміло кожному інженеру: чим більше даних, тим вище обчислювальна складність. Поки ми працюємо з наборами іграшкових даних (набори, що створені спеціально для академічних цілей), їх розмір не є проблемою, але для реальних систем сотні додаткових ознак будуть створювати відчутний дискомфорт. Друга причина полягає в тому, що деякі алгоритми дуже чутливі до так-званого шуму (не інформативних ознак). Можна проілюструвати цей феномен, використавши, як приклад, задачу кредитного скорінгу (давати чи не давати людині кредит) та алгоритм дерева рішень. Під час передбачення дерево рішень розбиває навчальні дані на класи, використовуючи ознаки, які ми надали. Наприклад, чи отримує клієнт більше двадцяти тисяч, чи є у клієнта автомобіль, та ін. Очевидно, що такі ознаки дійсно впливають на те, чи поверне клієнт кредит, чи ні. Однак, якщо б ми надали зайві ознаки, наприклад, якого кольору шкарпетки полюбляє людина, це тільки б заплутало наш алгоритм.

Статистичні підходи до проблеми. Найбільш очевидним кандидатом на видалення є ознака, значення якої залишається незмінною, тобто вона взагалі не містить інформації. Якщо спиратися на цю думку, доцільно сказати, що функції з низькою дисперсією гірші, ніж ті, що мають велику дисперсію. Отже, можна просто відкидати ознаки з дисперсією нижче якогось порогу. Такий простий спосіб і є найпростішим методом відбору ознак. Існують також підходи на основі більш складних статистичних характеристик, таких як, наприклад, критерій хи-квадрат.

Відбір моделюванням (англ. selection by modeling). Підхід полягає у використанні деякої базової моделі для оцінки ознак, оскільки модель чітко покаже їх важливість. Зазвичай використовують два типи моделей: на основі дерев рішень, такі як «випадковий ліс», або лінійну модель з регулярізацією Лассо, щоб вона зводила до нуля ваги слабких ознак. Логіка є інтуїтивно зрозумілою: якщо ознаки погано показують себе у простій моделі, немає необхідності перетягувати їх на більш складну. Використання випадкового лісу іноді особливо виправдане, адже ця модель дозволяє вивести графік важливості ознак за спаданням.

Також популярними є алгоритми, які основані на різних версіях **повного або часткового перебору** всіх можливих комбінацій ознак. Одним з них є алгоритм рекурсивного усунення ознак. Він полягає у виборі ознак шляхом рекурсивного врахування все менших та менших їх наборів. По-перше, оцінювач навчається на початковому наборі ознак, та отримується важливість кожної з них. Тоді найменш важливі ознаки відкидаються від поточного набору. Ця процедура рекурсивно повторюється на зменшеному наборі, поки в кінцевому підсумку не буде досягнуто оптимальної кількості ознак. Очевидно, що таким чином ми переберемо далеко не всі комбінації, однак, вичерпний метод перебору є дуже витратним з точки зору часу та ресурсів. Якщо ж, все-ж таки потрібно перебрати багато варіантів, то зазвичай роблять так: фіксують невелике число N, перебирають всі комбінації з N ознак, обирають найкращу комбінацію, а потім повторюють перебір комбінацій проте вже з N + 1, так щоб попередня найкраща комбінація була зафіксована, і лише одна нова ознака перебирається. Таким чином можна повторювати, поки в нас не закінчаться ознаки або поки якість моделі не перестане значно зростати. Цей алгоритм називається послідовним вибором функцій (англ. Sequential Feature Selection).

2.5 Зменшення розмірності

Очевидно, при простому відкиданні малоінформативних ознак, повністю втрачається вся інформація, яка в них знаходилась. Проте, це не завжди вигідно, часто, розробник просто хоче зменшити кількість ознак у даних, для пришвидшення роботи моделі або задля їх візуалізації, без втрати великої кількості інформації. У таких випадках використовують алгоритми **зменшення розмірності.** Цікава аналогія приводиться в [2]. Автор розглядає процес зменшення розмірності, як споріднений до стиснення зображень. Всі, хто колись зіштовхувались з цим процесом, знають, що зображення може бути стиснене в багато разів (інколи в десятки), але помітити різницю неозброєним оком все-одно важко. Наприклад, алгоритм jpeg60 стискає (зменшує розмір) зображення у 17 разів, проте зберігає близько 60% інформації, чого абсолютно вистачає для повсякденного життя (різниця в якості майже не помітна). Мета збереження якомога більшого відсотку інформації при зменшення розмірності ставиться і в машинному навчанні.

Найпопулярнішим способом зменшення розмірності є метод головних компонент (англ. **principal component analysis**) [12]. PCA визначає вісь, на яку припадає найбільша кількість дисперсії у навчальному наборі. На рисунку 2.5 це суцільна лінія. Він також знаходить другу вісь, ортогональну першій, на яку припадає найбільша кількість дисперсії, що залишилася. У цьому двовимірному прикладі вибору немає: це пунктирна лінія. Якби це був набір даних з більшою розмірністю, PCA також знайшов би третю вісь, ортогональну обом попереднім осям, і четверту, п’яту тощо - стільки осей, скільки кількість вимірів у наборі даних. Одиничний вектор, що визначає *i-ту* вісь, називають *i-тим* головним компонентом. На рисунку 2.5 ними є c1 та c2 відповідно. Якщо б в нас були дані з розмірністю 3, третя головна компонента була б ортогональна до c1 та c2. Тож як можна знайти головні компоненти навчального набору? Існує стандартна методика факторизації матриць під назвою Singular Value Decomposition (SVD), яка може розкласти

матрицю навчального набору X на скалярний добуток трьох матриць $U \cdot \Sigma \cdot V^T$, де V^T містить усі головні компоненти. Визначивши всі головні компоненти, ви можете зменшити розмірність набору даних до d, спроектувавши його на площину, базис якої визначений першими d головними компонентами. Вибір цієї площини гарантує, що проекція збереже якомога більше дисперсії. Щоб спроектувати навчальний набір на цю площину, потрібно помножити матрицю векторів ознак X на матрицю перших d головних компонент W (першій d колонки матриці V^T). Формула: $X_{poj} = X \cdot W_d$

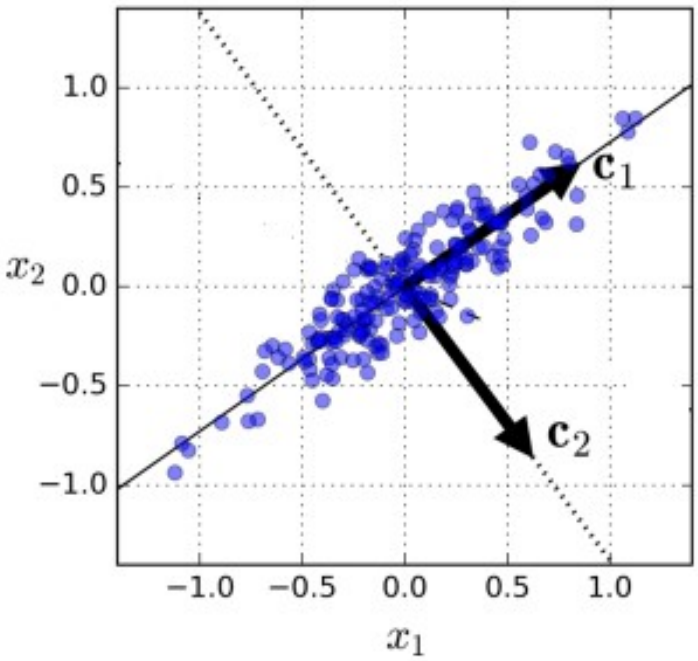


Рис 2.5 – Приклад головних компоненти.

2.6 Способи валідації

Найважливішим у кожній моделі машинного навчання звісно є те, наскільки добре ця модель узагальнюється (якість передбачення на нових прикладах). Оскільки ми не можемо відразу перевірити працездатність моделі на нових вхідних даних (оскільки ми ще не знаємо справжніх значень цільової змінної), необхідно пожертвувати невеликою частиною даних для перевірки якості моделі на ній [13]. Набір даних на якому буде здійснюватися фінальна перевірка якості моделі називається – тестовим, його відкладають

на самому початку роботи над проектом і більше ніколи не чіпають аж до самої фінальної перевірки якості. Тестовий набір повинен бути даними які

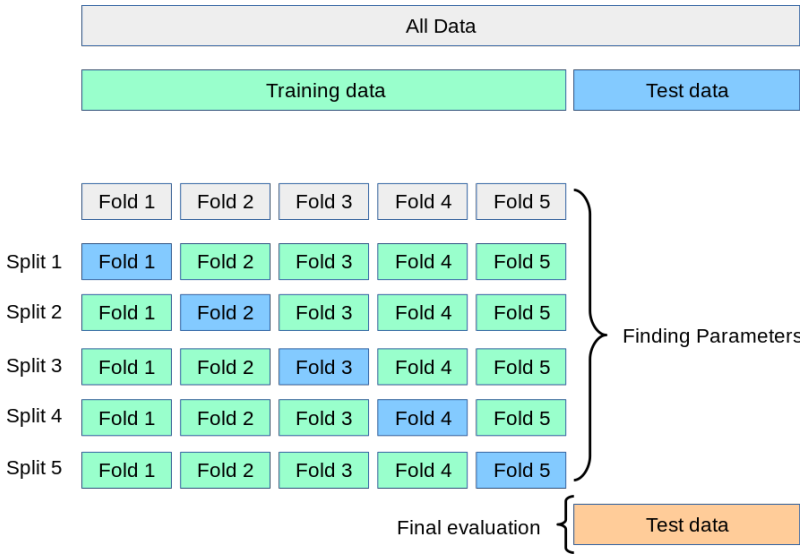


Рис 2.6 – Приклад правильного розподілу даних

модель дійсно ніколи не бачила. Частину даних, що в нас залишилась потрібно розбити ще раз, на одній частині буде відбуватися власне навчання моделі (підбір параметрів самого алгоритму шляхом мінімізації функції помилки), а на іншій – підбір так-званих гіперпараметрів (параметри моделі які обирає та налаштовує сам розробник).

Є дві популярні методики такого розбиття:

- Відкласти частину навчального набору даних (hold-out set). Як правило, відкладається від 20% до 40%, потім модель навчається на решті даних (60-80% від початкового навчального набору). Далі робиться передбачення на відкладеному наборі, результати цього передбачення потім порівнюються із реальними значеннями цільової змінної (вони нам відомі). Маючи реальні та передбачені значення міток ми можемо оцінити роботу моделі, використавши будь-яку з наявних метрик якості.
- Використати методику перехресної перевірки (англ. cross-validation) (рис 2.6). У K-fold перехресній перевірці модель навчається K разів на різних (K − 1) підмножинах навчального набору даних (зеленим

кольором) та перевіряється на решті підмножини (кожного разу інший, показаний нижче блакитним кольором). Ми отримуємо оцінки якості K моделей, які зазвичай усереднюються, щоб дати загальну середню якість класифікації / регресії. Перехресне підтвердження забезпечує кращу оцінку якості моделі на нових даних порівняно з підходом відкладеного набору. Однак перехресна перевірка обчислювально дорога, коли у вас багато даних або складній алгоритм навчання.

2.7 Опис роботи моделей (алгоритмів) машинного навчання

Алгоритмів машинного навчання є дуже багато, постійно з'являються нові реалізації та ідеї. Детально описати всі в одній роботі дуже складно, тому в цьому підрозділі будуть описані лише ті, що є найпопулярнішими або були застосовані у цій роботі.

2.7.1 Лінійна регресія

Лінійна регресія [14] – популярний алгоритм машинного навчання, який вивчає модель, що є лінійним поєднанням особливостей вхідного прикладу. Алгоритм є найстарішим та, напевно, найпопулярнішим у всьому регресійному аналізі.

Постановка задачі була описана у розділі 1, проте нагадаємо її ще раз. Наявна колекція розмічених прикладів $\{(x_i, y_i)\}_{i=1}^{N}$. Кожен x_i з N є вектором ознак (англ. feature vector). Вектор ознак – це такий вектор у якому кожен його вимір $j=1,...,D$ містить значення, що якось описує приклад. Це значення називають ознакою (англ. feature) та позначають x^j. Для усіх прикладів у наборі, *j-ma* ознака з вектору ознак завжди містить однаковий вид інформації (наприклад вагу в кг). Мітка (англ. label) y_i може належати реальних чисел.

Ми будуємо модель $f_{(w,b)}(x)$ як лінійну комбінацію ознак прикладу x:

$f_{(w,b)}(x) = wx + b$, де w це D-розмірний вектор параметрів, а b - дійсне число. Позначення $f_{(w,b)}(x)$ означає, що модель f параметризована двома значеннями: w і b. Якщо ми робимо наші передбачення лише за однією ознакою, то нашу

модель та приклади можна зобразити на звичайному графіку (рис 2.7). На рисунку ми бачимо, що наша модель передбачень – це звичайна пряма, що задається рівнянням $y=wx+b$, а тренувальні приклад це просто точки на площині. Очевидно, що метою є лінія, яка проходить якомога ближче до тренувальних прикладів (точок), адже це означає, що модель гарно описує тренувальні, а значить і нові дані(робиться припущення, що нові приклади будуть схожими на тренувальні).

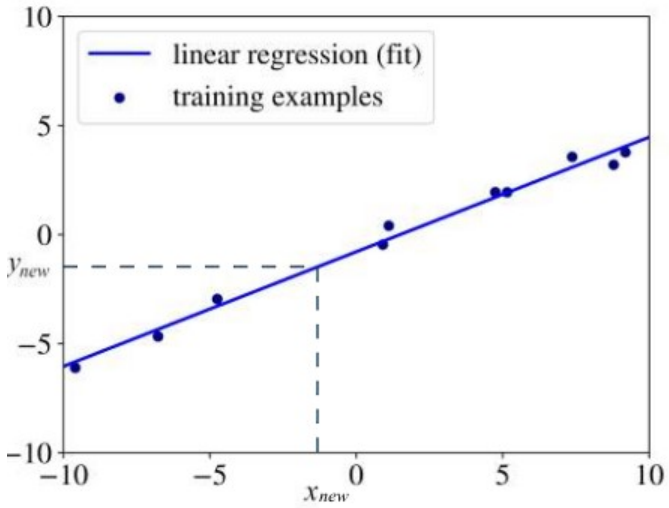


Рис 2.7 – Лінійна регресія.

Щоб задовольнити мету, поставлену вище, ми використаємо ітераційний алгоритм оптимізації (наприклад градієнтний спукс), щоб знайти такі оптимальні значення w^* та b^* , які мінімізують вираз : $\frac{1}{2N}\sum_{i=1\ldots N}\left(f_{|w,b}(x_i)-y_i\right)^2$. Функцію яку мінімізує алгоритми оптимізації називають цільовою функцією. Вираз $(f_{(w,b)}(x_i)-y_i)^2$ визначається як функція втрат. Це міра штрафу за неправильну відповідь стосовно i-го прикладу. Цей конкретний вибір функції втрати називається квадратичною функцією втрат. Отже цільовою функцією у цьому випадку є середньоквадратична похибка. Середня похибка, або емпіричний ризик для моделі, є середній показник усіх штрафів, отриманих протягом застосуванням моделі до навчальних даних. Вибір саме лінійної комбінації ознак та квадратичної функції втрат пояснюється здебільшого стійкістю такої системи до перенавчання та легкістю оптимізації.

2.7.2 Логістична регресія

Спершу, треба зазначити, що не зважаючи на назву, алгоритм є не частиною регресійного аналізу, а одним з алгоритмів класифікації. Таке схожі назви були дані алгоритмам через схожість математичних концепцій на яких вони збудовані. У цьому підрозділі буде наведене пояснення роботи алгоритму з точки зору бінарної класифікації, проте принципи роботу залишаються незмінними і для мультикласового варіанту.

Постановка задачі є ідентичною минулому підрозділу з тією різницею, що y_i належить одній з двох дискретних категорій [14]. Проблема закладається в тому, що лінійна комбінація ознак $wx_i + b$ може мати значення від мінус до плюс нескінченності, тоді як мітка y_i повинна належати одній з двох категорій. Визначимо негативну мітку як 0, а позитивну мітку як 1, тоді нам просто потрібно знайти просту безперервну функцію, область значень якої є (0, 1). У такому випадку, якщо значення, повернене моделлю після введення ознак x, ближче до 0, тоді ми присвоюємо негативну мітку; в іншому випадку приклад позначений міткою як позитивний. Однією з функцій (рис 1.15), яка має таку властивість, є стандартна логістична функція (також відома як сигмоїда): $f(x) = \frac{1}{(1 + e^{-x})}$, де e це число Ейлера.

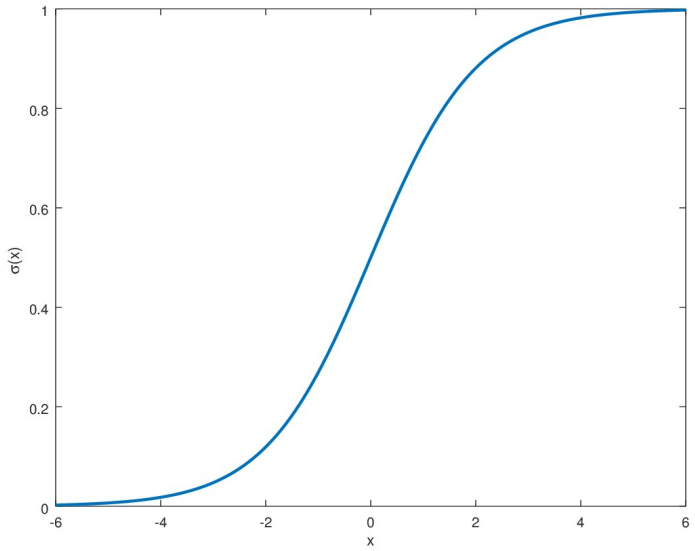


Рис 2.8 – Стандартна логістична функція.

На рисунку 2.8, ми можемо побачити, наскільки добре відповідає нашій цілі класифікації така функція: якщо ми оптимізуємо значення *w* і *b* відповідним чином, ми могли б інтерпретувати результат *f(x)* як ймовірність отримання позитивної мітки. Наприклад, якщо він вище або дорівнює порогу 0.5, ми б сказали, що клас прикладу *x* позитивний; інакше – негативний (значення порогу обирає розробник виходячи з обставин конкретної прикладної задачі). Отже модель логістичної регресії виглядає так: $f(x) = \frac{1}{(1 + e^{-(wx+b)})}$. Лінійна комбінація ознак *wx+b* нічим не відрізняється від використаної в лінійній регресії.

При логістичній регресії замість того, щоб мінімізувати середньоквадратичну похибку, ми максимізуємо так-звану функцію правдоподібності (максимальної вірогідності). Припустимо, в нас є розмічений приклад та ми вже знайшли параметри *w* і *b.* Тепер, при поданні вектора ознак нашого прикладу до моделі, отримаємо відповідь p, що буде лежати в межах: *0 < p < 1*. Отже, згідно нашої моделі, *p* це вірогідність того, що мітка прикладу є позитивною, *1 - p*, відповідно – негативною. Максимізація функції правдоподібності є максимізацією впевненості моделі у правильності її відповідей. З причин математичної зручності за цільову функцію прийнято брати логарифм функції правдоподібності:

$$LogL_{\mathbf{w},b} \stackrel{\text{def}}{=} \ln(L(_{\mathbf{w},b}(\mathbf{x})) = \sum_{i=1}^{N} y_i \ln f_{\mathbf{w},b}(\mathbf{x}) + (1 - y_i) \ln (1 - f_{\mathbf{w},b}(\mathbf{x})).$$

У якості алгоритму оптимізації зазвичай використовують градієнтний спуск.

2.7.3 Проблема перенавчання

Дивлячись на описи більшості алгоритмів машинного навчання ми бачимо що їх головною цілю є зменшення похибки класифікації або регресії на навчальних даних (навчальному наборі). Це загалом правильна тактика, адже ми робимо припущення, що нові дані будуть схожими на навчальні. Однак, що буде коли ми зведемо цю похибку до нуля? Здавалося б це

прекрасний новина, наша модель взагалі не помиляється на тренувальних даних, значить не буде і на нових? На жаль, це зовсім не так. Практика показує, що коли похибка моделі близиться до нуля, її дисперсія стрімко збільшується (рис. 2.9). Спрощено, ми можемо сказати, що модель “завчила” всі тренувальні приклади, проте, нові приклади будуть просто схожими, а не ідентичними до тренувальних, і вже на них модель не зможе продемонструвати таких гарних результатів. Тобто модель починає погано узагальнюватись. Цей феномен прийнято називати – перенавчанням.

Звісно, це зовсім не означає, що ми не повинні прагнути зменшення похибки. Ситуацію коли похибка моделі є завеликою називають – недонавчанням. Просто зменшуючи похибку ми постійно повинні перевіряти модель на нових даних (способи перевірки будуть описані пізніше), щоб зрозуміти чи не почалося перенавчання. В ідеалі ми б хотіли отримати модель з мінімальною похибкою (зсувом) та мінімальною дисперсією. Проте, на жаль, зазвичай це неможливо і розробникам приходиться вирішувати задачу компромісу зсуву та дисперсії (англ. bias–variance tradeoff) (рис 1.16).

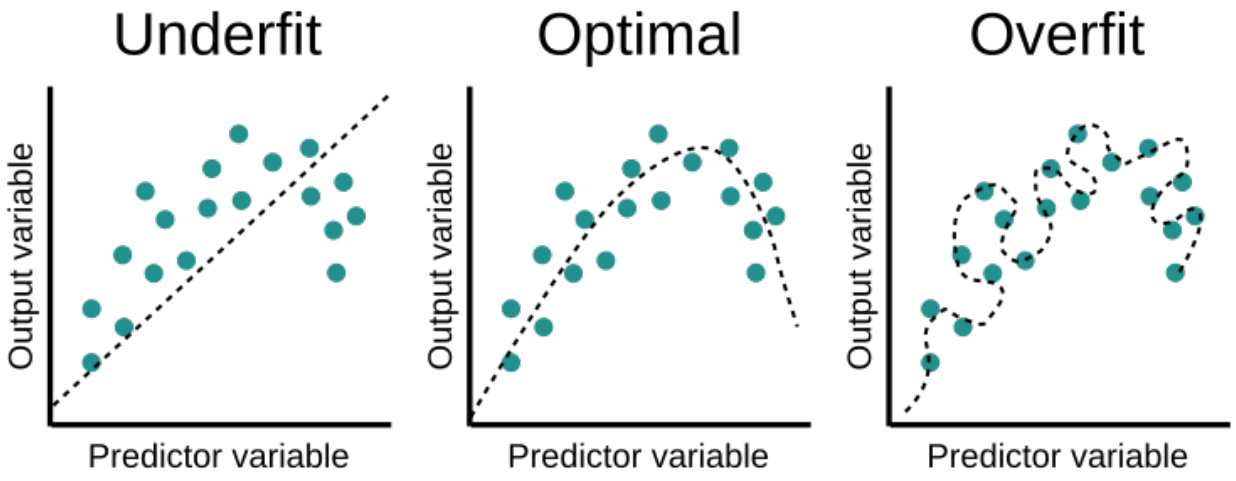


Рис 2.9 – Приклад компромісу зсуву та дисперсії.

Методи зменшення зсуву (боротьби з недонавчанням) є інтуїтивними.

- Конструювання більшої кількості (або більш інформативних) ознак.
- Обрати більш складну модель, наприклад, метод опорних векторів замість логістичної регресії.

Існують також і методи боротьби з перенавчанням:

- Обрати більший за розміром навчальний набір (однак, часто це неможливо).
- Зменшити розмірність тренувальних даних (наприклад за допомогою методу головних компонент або за допомогою відбору ознак)
- Обрати простішу модель.
- Застосувати регуляризацію.

Саме **регуляризація** є пріоритетним варіантом, оскільки інші зазвичай або неможливі, або призводять до збільшення похибки.

2.7.4 Регуляризація

Регуляризація [13] – це загальний термін, який охоплює методи, що змушують алгоритм протягом навчання будувати менш складні моделі. На практиці це часто призводить до дещо більшого зсуву, але значно зменшує дисперсію. Найпопулярнішими методами є L1 та L2 регуляризація. Щоб створити регуляризовану модель, потрібно її функції втрат додати спеціальний регуляризуючий вираз. Значення цього виразу збільшується в міру зростання складності моделі. Регуляризуючі вирази можуть бути додані до великої кількості моделей, проте найлегше продемонструвати на прикладі лінійної регресії. Регресія з доданим виразом L1 називається – лассо регресія (англ. Lasso Regression), а з виразом L2 – регресія хребта (англ. Ridge Regression).

Регресія хребта додає до функції втрат штрафний вираз у вигляді квадрату величини параметрів: $\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$, якщо lambda буде рівною нулю, то отримуємо стандартний вигляд функції втрат лінійної регресії. Проте, якщо lambda буде не нульовою, то алгоритм оптимізації буде намагатися не робити параметри завеликими, адже він буде за це штрафуватися. Отже, чим більша lambda, тим суворіші обмеження на значення параметрів, тим простіша модель, тим більша регуляризація.

Зрозуміло, що ми не хочемо отримати надто просту модель, адже це призведе до недонавчання. Значення lambda є одним з параметрів які обирає сам розробник, виходячи з обставин конкретної задачі (так звані гіперпараметрт).

Лассо регресія має схожий принцип роботи, проте формула штрафного виразу дещо інша та є абсолютним значенням величини параметрів моделі: $\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$. Роль гіперпараметра lambda не змінилася. Ключова відмінність цих методів полягає в тому, що Лассо зменшує коефіцієнт менш важливої ознаки до нуля, тим самим повністю відкидаючи її. Таким чином, метод добре підходить для вирішення проблеми відбору ознак. Гіперпараметр lambda зазвичай обирається шляхом часткового перебору можливих значень та перевірки результатів роботи моделі на нових даних з цими значеннями, використовуючи перехресну перевірку.

Існує також варіація лінійної регресії під назвою **Elastic Net**, що має на меті поєднати властивості обох попередніх варіантів. У якості штрафного виразу у цій варіації виступає сума L1 та L2 регуляризаційних виразів. Замість одного параметра lambda Elastic Net використовує два, кожен з яких регулює степінь впливу L1 та L2 виразів відповідно.

2.7.4 Дерево ухвалення рішень

Дерево ухвалення рішень є популярним алгоритмом машинного навчання з учителем, що може використовуватись як в задачах класифікації так і регресії [13]. Однак, основні концепції та підходи, що лежать в основі цього алгоритму, використовуються людиною і в повсякденному житті. Діаграми потоків даних (рис 2.10), що є візуалізацією дерева рішень, використовуються далеко за межами проблем математичного моделювання. Саме легка інтерпретація алгоритму через такі діаграми призвела до росту його популярності.

Дерево на рисунку 2.10 є дуже простим і може бути побудоване взагалі без використання машинного навчання. Натомість можна звернутися до

експертів (у цьому випадку до експертів з гри в гольф), які самі напишуть ланцюжки правил за якими буде дана відповідь (чи можна сьогодні грати в

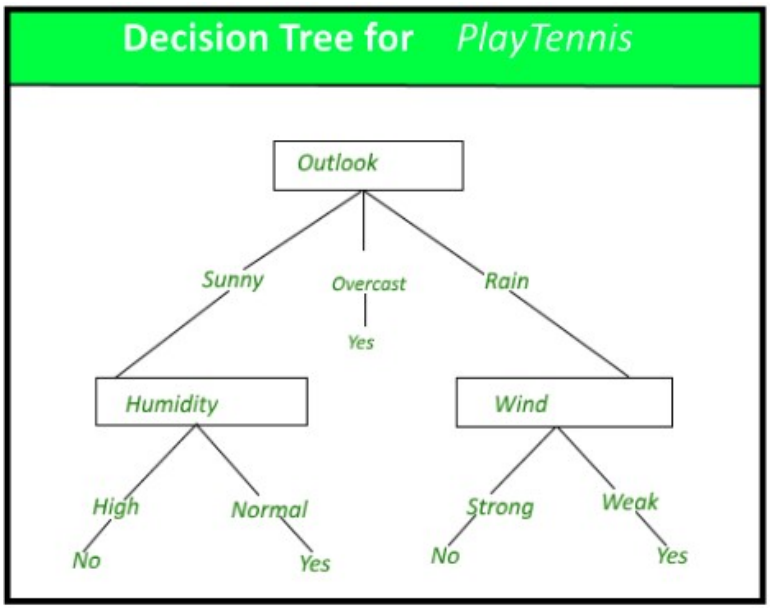


Рис 2.10 – Приклад візуалізації дерева рішень

теніс), виходячи зі свого досвіду. Однак, часто, проблеми, що стоять перед розробниками є набагато складнішими (наприклад, кредитний скорінг) і написання правил експертами є дуже довгим, дорогим, а іноді й нездійсненим завданням. Автоматизоване створення таких правил і називають навчанням у випадку дерева рішень.

Дерево рішень як алгоритм машинного навчання по суті перетворює потік логічних правил форми "значення ознаки *a* менша за *x*, а значення ознаки *b* більше за *y* ... => категорія 1" в структуру даних, що нагадує дерево. Щоб алгоритм побачив, яку ознаку обрати, та де її слід розділити (сформувати правило) використовується концепція отримання інформації на основі ентропії. Ентропія Шеннона визначена для системи з можливою N так:

$S = -\sum_{i=1}^{N} p_i \log_2 p_i$, де p-і це ймовірність знаходження системи в і-тому стані. Ентропію можна охарактеризувати як ступінь хаосу в системі. Чим вище ентропія, тим менше впорядкована система і навпаки. Отже, “хороше” правило сильно зменшить ентропію системи. Приклади “хороших” та “поганих” правил можна побачити протягом гри в “20 запитань”. Перша

людина загадує знаменитість, а друга намагається відгадати, задаючи лише питання на які можна відповісти "Так" або "Ні". Яка тактика буде виграшною для другого гравця? Звичайно, ставити запитання, що найбільше звужують коло можливих варіантів. Запитання "Це Тарас Шевченко?" у випадку негативної відповіді залишить усі можливі варіанти крім однієї знаменитості. На противагу цьому, запитання "Чи знаменитість жінка?" зменшує кількість варіантів приблизно у два рази. Тобто ознака "стать" розділяє набір знаменитостей набагато краще, ніж інші ознаки, наявне значне зменшення ентропії. Замість ентропії можуть бути використані інші критерії, наприклад джині але загальні принципи є однаковими. Кількість питань (глибина дерева) та їх специфічність (скільки варіантів залишається в кожній категорії після розбиття) є гіперпараметрами, їх неправильний вибір може призвести до перенавчання.

2.8 Оцінка якості передбачень моделі

Як під час процесу валідації (вибір моделі та гіперпараметрів), так і для остаточної оцінки якості моделі на тестовій вибірці необхідно використовувати одну з метрик оцінки якості передбачень. Оскільки у цій роботі стоїть задача класифікації далі будуть розглянуті метрики для оцінки саме її якості.

Звісно, найпростішим мірилом якості є проста **доля (відсоток)** правильно класифікованих прикладів. Однак, якщо розподіл цільової ознаки є нерівномірним, використання такої метрики одразу втрачає сенс. Наприклад якщо наявні 90 позитивних та 10 негативних міток, то класифікатор може завжди присвоювати прикладам позитивну мітку та буде мати при цьому 90% правильних відповідей. Зрозуміло, що попри гарне значення цієї метрики, модель нічому не навчилася, а її корисність є нульовою. Для незбалансованих наборів використовують інші, більш досконалі, метрики якості.

Першим варіантом такої вдосконаленої метрики є **матриця помилок** (англ. **confusion matrix**). Матриця помилок – це таблиця, яка підсумовує,

наскільки успішна модель класифікації при прогнозуванні прикладів різних класів. Одна вісь матриці – це мітки, які передбачила модель, а інша вісь - фактичні мітки. Наприклад, є двійковій проблема класифікації, модель передбачає два класи: "спам" та "не_спам", тоді матриця помилок буде виглядати так:

	“спам”(передбачено)	“не_спам”(передбачено)
“спам”(фактично)	23(TP)	1(FN)
“не_спам”(фактично)	12(FP)	556(TN)

Табл. 2.1 – Приклад матриці помилок.

У цьому прикладі з 24-ох спам прикладів система коректно класифікувала 23. Іншими словами, маємо 23 правильно спрогнозованих позитивних результати (англ. true positives, TP) та 1 неправильно (анг. false negative, FN). Так само і з негативними прикладами: 556 правильно спрогнозованих (англ. true negative, TN), а 12 неправильно (анг. false positives, FP).

Неправильно спрогнозовані позитивні та негативні приклади прийнято називати помилками першого роду та другого роду відповідно. Така матриця для задачі мультикласової класифікації буде мати більше рядків та стовпців проте метод побудови буде незмінним. Хоча матрицю помилок можна використовувати для власне оцінки якості передбачень, частіше за її допомогою розраховують значення двох інших метрик: точність та повноту.

Точність [14] – це відношення правильних позитивних прогнозів до загальної кількості позитивних прогнозів: $Precision = \frac{TP}{TP + FP}$. **Повнота** [14] – це відношення правильних позитивних прогнозів до загальної кількості позитивних прикладів у наборі даних: $Recall = \frac{TP}{TP + FN}$. У випадку, наприклад, спам-фільтру зазвичай робиться наголос на точності, адже дуже критично, щоб важливий лист не був помічений як спам. Заради високої точності часто приходиться жертвувати повнотою та навпаки. Комбінацією метрик повноти та точності є **F1-міра** [14], вона визначається як: $F_1 = 2 \times \frac{precision \times recall}{precision + recall}$. З

формули можна побачити, що значення міри буде високим (близьким до 1), тільки якщо значення і точності, і повноти будуть високими (близькими до 1).

Популярною метрикою також є площа, під **ROC-кривою** (англ. Area Under ROC Curve, **AUC-ROC**) [14]. ROC-крива відображає співвідношення повноти (англ. true positive rate, TPR) до частки помилкових позитивних класифікацій (англ. false positive rate, FPR). Чим більша площа під ROC-кривою (рис 2.11), тим якісніше діє класифікатор, при цьому значення 0,5 демонструє непридатність обраного методу класифікації (відповідає звичайному вгадуванню)[https://loginom.ru/blog/logistic-regression-roc-auc].

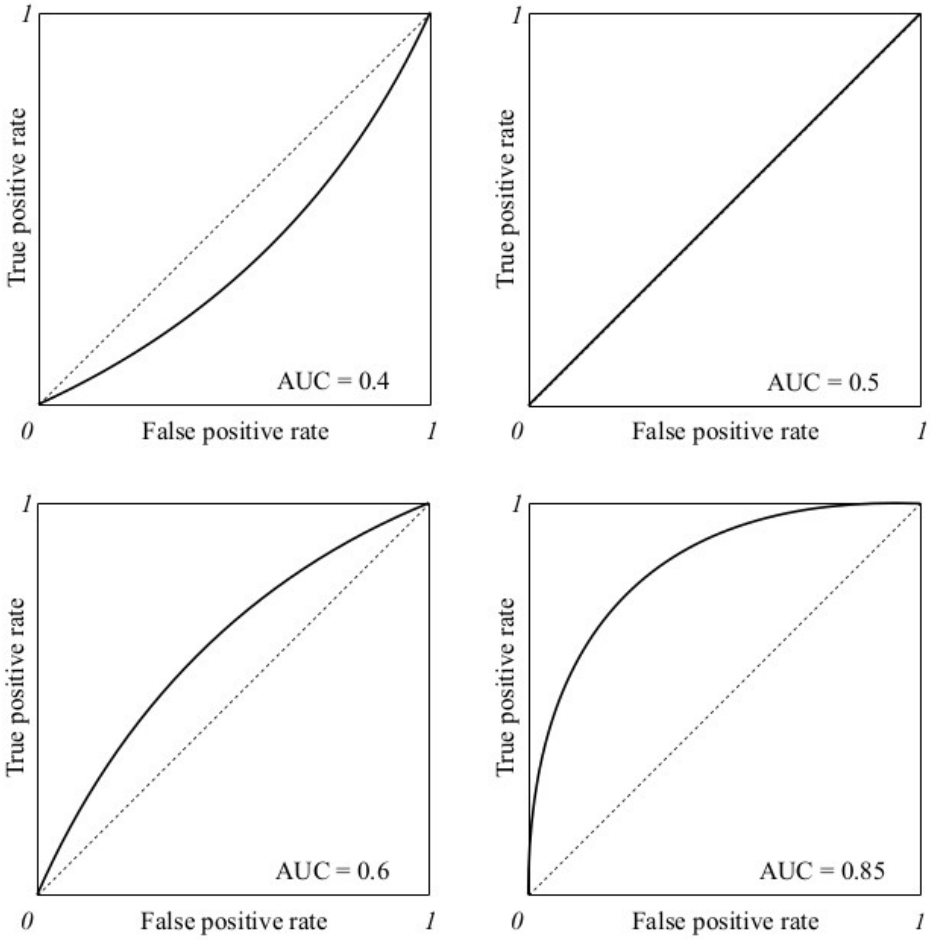


Рис. 2.11 – Приклади визначення AUC-ROC.

Усі перераховані вище метрики є дуже популярними та вживаними, проте вони мають один спільний недолік: немає диференціювання неправильних класифікацій. У випадку цієї дипломної роботи, якщо система видасть твору оцінку 4 бали, а фактична оцінка 5 балів, приклад буде

вважатися повністю неправильно класифікованим. Зрозуміло, що така тактика оцінки якості роботи є абсолютно не продуктивною. Саме в таких випадках використовують метрики, що вимірюють **згоду** оцінок двох експертів, які оцінюють одні і ті ж приклади. До таких метрик входять: Cohen's kappa, Scott's pi, Fleiss' kappa, та інші. Оскільки саме каппа Коена є найпопулярнішою, вона буде розглянута більш детально.

Квадратично зважена каппа Коена (англ. Quadratic Weighted Cohen's kappa) – це міра згоди між двома оцінювачами, що бере до уваги випадковий збіг оцінок. Припустимо в нас є набір даних де цільова зміна належить одному з чотирьох класів 0-4. Тоді для отримання цієї метрики необхідно пройти такі кроки [15]:

1. Побудувати мультикласову матрицю помилок *O*.
2. Створити матрицю вагів *W*, щоб передбачення, які знаходяться далі від фактичного значення важили менше.
3. Створити два нових вектора які описують скільки значень кожного класу наявні в векторах фактичних та передбачених значень цільової змінної.
4. Створити матрицю *E*, що є тензорним добутком двох векторів описаних у минулому кроці.
5. Нормалізувати обидві матриці, щоб вони мали однакову суму (наприклад, поділити кожну на суму її елементів, тоді обидві матриці будуть мати суму 1).
6. Зробити остаточні обчислення за формулою:

$$WeightedKappa = 1 - \frac{\sum_{i,j=0}^{len(W)} W_{(i,j)} * O_{(i,j)}}{\sum_{i,j=0}^{len(W)} W_{(i,j)} * E_{(i,j)}}.$$

Значення метрики лежать в діапазоні від -1 до 1, де -1 – це повна відсутність згоди, 0 – згода що близька до випадкового збігу, а 0.6+ – відповідає за істотну згоду між оцінювачами.

Висновки до розділу 2

У цьому розділі були описані методи попередньої обробки тексту та представлення його у вигляді придатному до опрацювання математичними засобами (у векторному вигляді); методи конструювання, відбору та нормалізації ознак. Була розглянута проблема перенавчання та способи її виріщення. Також були проаналізовані такі алгоритми машинного навчання як: Лінійна, Логістична, Ласо (англ. Lasso) та Хребтова (англ. Ridge) регресія; Дерево рішень; популярні способи валідації та оцінки якості роботи моделей.

РОЗДІЛ 3
РОЗРОБКА ОБРАНОЇ МОДЕЛІ КЛАСИФІКАТОРА

3.1 Обраний набір даних

Як вже зазначалося, машинне навчання з учителем великою мірою залежить від об'єму та якості даних. Більш того, у реальних задачах, з популяризацією моделей на основі концепції "чорного ящику", вибір та підготовка даних починає забирати навіть більше часту та зусиль ніж вибір розробка та тестування математичних алгоритмів. У нашому випадку, пошук великого набору есе, який можна було б використати в дослідженні, був зовсім не простим завданням. Велика кількість якісних наборів тренувальних та тестових даних можна знайти на сайті Kaggle.com. Kaggle – це платформа, яка спеціалізується у сфері наук про дані та прогнозованого моделювання. Численні організації розміщують свої дані на цій платформі, щоб привернути увагу дослідників з усього світу, які могли б знайти цікаві залежності в даних або зробити якісь передбачення. Набір даних, який використовується у цій роботі також був узятий з цього ресурсу, він називається "The Hewlett Foundation: Automated Essay Scoring DataSet". Як ми можемо побачити з назви, створення цього набору було спонсовано благодійним фондом Вільяма та Флори Хюлетт, також відомий як "The Hewlett Foundation". На жаль, це набір творів, написаних англійською, знайти датасет такої якості українською не вдалося.

Після видалення прикладів з пропусками хоча б в одній з колонок та не інформативної ознаки унікального ідентифікатора кожного прикладу, ми маємо такий набір даних: приблизно 12 тисяч прикладів есе, написаних учнями починаючи з сьомих класів; всі приклади есе поділені на 8 типів (підгруп). Набір даних має такі початкові поля:

- essay_set – до якого з 8-и типів належить приклад
- essay – есе у вигляди тексту

- domain1_score – оцінка, що була поставлена есе

Кожна з 8-и підгруп на які поділені дані відповідає за окремий тип есе. В кожної підгрупи своя шкала оцінок. Умовно, можна поділити ці типи на два класи: есе на вільну тему та есе за прочитаним твором.

На рисунку 3.1, можемо побачити як виглядають дані, а на рисунку 3.2 розподіл прикладів за типами. Також на рисунку 3.3 проілюстрований розподіл оцінок за різними шкалами відповідно до типу.

	essay_set	essay	domain1_score
0	1	Dear local newspaper, I think effects computer...	8.0
1	1	Dear @CAPS1 @CAPS2, I believe that using compu...	9.0
2	1	Dear, @CAPS1 @CAPS2 @CAPS3 More and more peopl...	7.0
3	1	Dear Local Newspaper, @CAPS1 I have found that...	10.0
4	1	Dear @LOCATION1, I know having computers has a...	8.0

Рис 3.1 – Приклад вигляду даних.

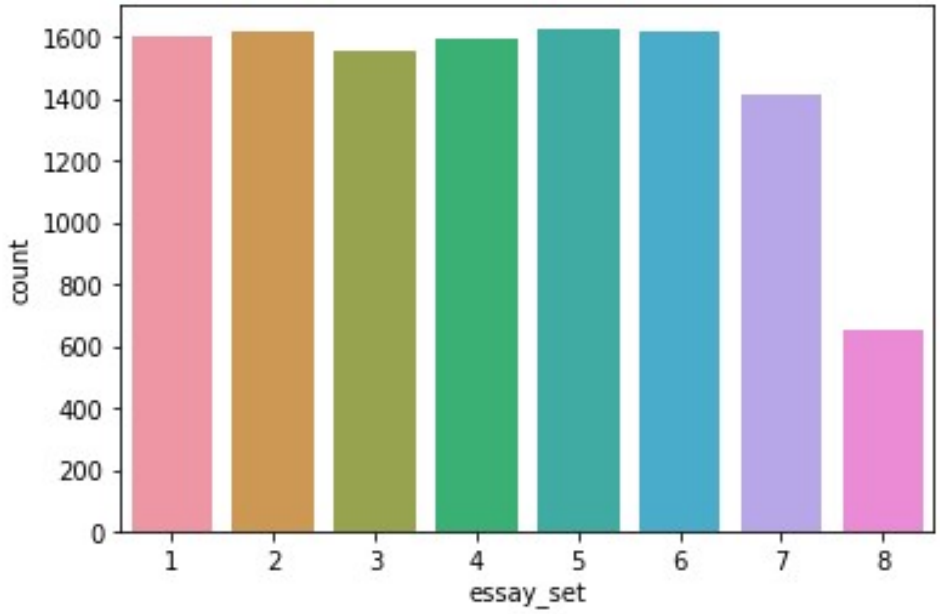


Рис. 3.2 – Розподіл прикладів за типами.

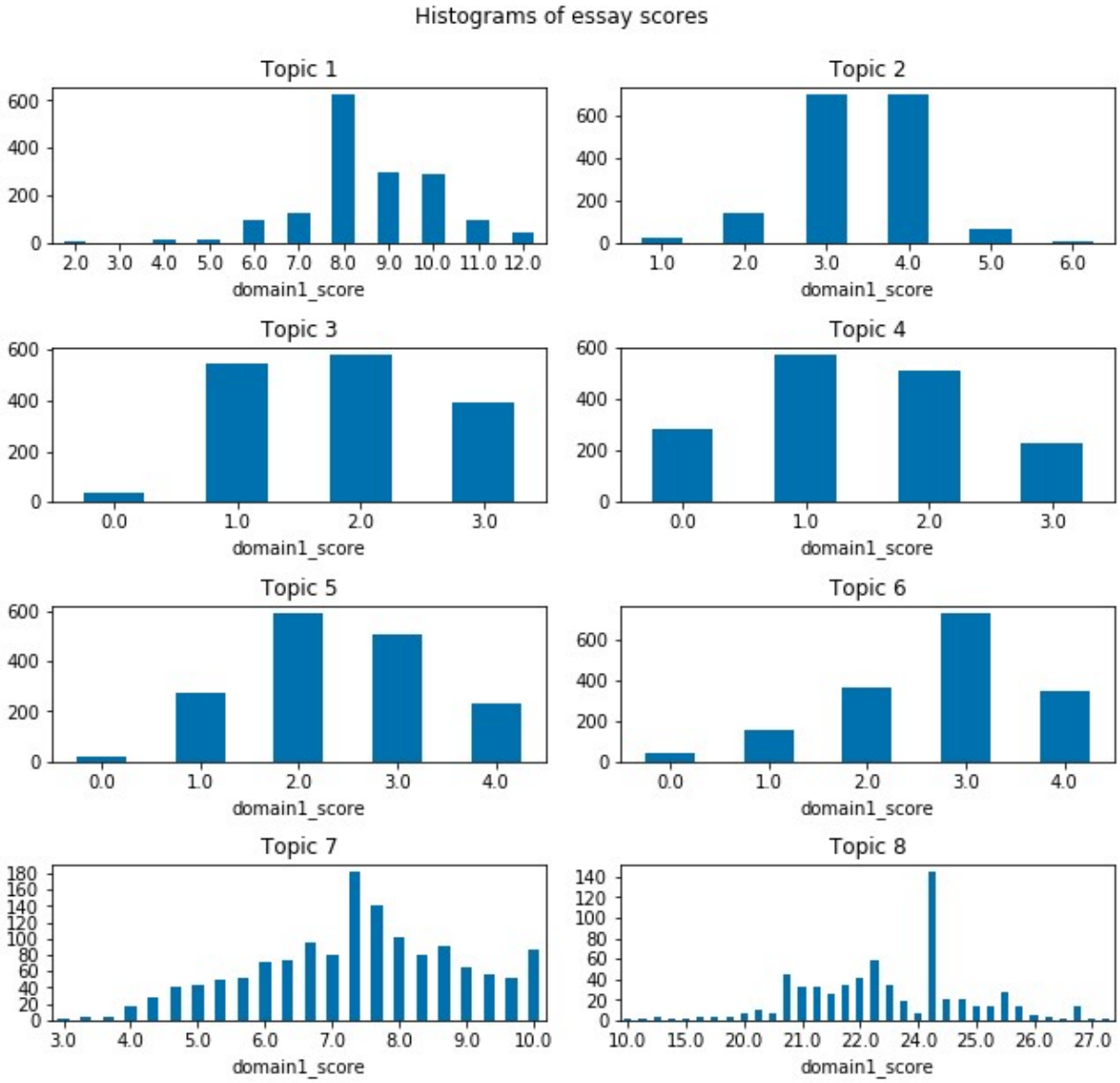


Рис 3.3 – Розподіл оцінок за різними шкалами відповідно до підгрупи.

3.2 Попередня обробка даних та конструювання числових ознак

У цій роботі попередня обробка даних була скомбінована з конструюванням числових ознак. Цей процес відбувався поетапно:

1. Зведення усіх слів в прикладах до нижнього регістру. Ознака – кількість символів у есе.
2. Токенізація кожного твору на окремі речення. Ознака – кількість речень в есе.
3. Видалення пунктуації. Ознака – кількість знаків пунктуації у есе.

4. Токенізація кожного есе на окремі слова. Ознаки: звичайна кількість слів, кількість унікальних слів, середня кількість слів в одному реченні та середня кількість символів у реченні.
5. Перевіряємо кожне слово на наявність помилок у написанні. Ознака – кількість орфографічних помилок в есе.
6. Рахуємо кількість слів англійської мови. Ознака – кількість слів англійською.
7. Видаляємо стоп-слова. Ознака – кількість стоп-слів
8. Визначаємо частину мови кожного слова. Конструюємо 40 ознак, кожна ознака описує кількість відповідної частини мови в есе.
9. Застосовуємо лематизацію та одразу конструюємо ознаку, що виражена в кількості лем.

Нижче наведений приклад текстового документу до та після попередньої обробки.

До обробки: "*Dear local newspaper, I think effects computers have on people are great learning skills/affects because they give us time to chat with friends/new people, helps us learn about the globe(astronomy) and keeps us out of troble! Thing about! Dont you think so? How would you feel if your teenager is always on the phone with friends! Do you ever time to chat with your friends or buisness partner about things. Well now - there's a new way to chat the computer, theirs plenty of sites on the internet to do so: @ORGANIZATION1, @ORGANIZATION2, @CAPS1, facebook, myspace ect. Just think now while your setting up meeting with your boss on the computer, your teenager is having fun on the phone not rushing to get off cause you want to use it. How did you learn about other countrys/states outside of yours? Well I have by computer/internet, it's a new way to learn about what going on in our time! You might think your child spends a lot of time on the computer, but ask them so question about the economy, sea floor spreading or even about the @DATE1's you'll be surprise at how much he/she knows. Believe it or not the computer is much interesting then in class all day reading out of books. If your child is home on your computer or at a local library,*

it's better than being out with friends being fresh, or being perpressured to doing something they know isnt right. You might not know where your child is, @CAPS2 forbidde in a hospital bed because of a drive-by. Rather than your child on the computer learning, chatting or just playing games, safe and sound in your home or community place. Now I hope you have reached a point to understand and agree with me, because computers can have great effects on you or child because it gives us time to chat with friends/new people, helps us learn about the globe and believe or not keeps us out of troble. Thank you for listening."

Після обробки: *'dear local newspaper think effect computer people great learning skill / affect time chat friend / new people help learn globe astronomy keep troble thing do not think feel teenager phone friend time chat friend buisness partner thing new way chat computer theirs plenty site internet organization1 organization2 caps1 facebook myspace ect think set meeting boss computer teenager have fun phone rushing cause want use learn countrys / state outside computer / internet new way learn go time think child spend lot time computer ask question economy sea floor spread date1 surprise -PRON- / -PRON- know believe computer interesting class day read book child home computer local library well friend fresh perpressured know be not right know child caps2 forbidde hospital bed drive - by child computer learn chat playing game safe sound home community place hope reach point understand agree computer great effect child give time chat friend / new people help learn globe believe keep troble thank listen'*

Кожен приклад тепер має 50 ознак. Нижче представлені приклади залежності цільової змінної від різних ознак. Можна побачити, що не всі ознаки впливають на оцінку однаково, наприклад ознака "середня кількість слів у реченні" впливає набагато менше за "кількість токенів в есе".

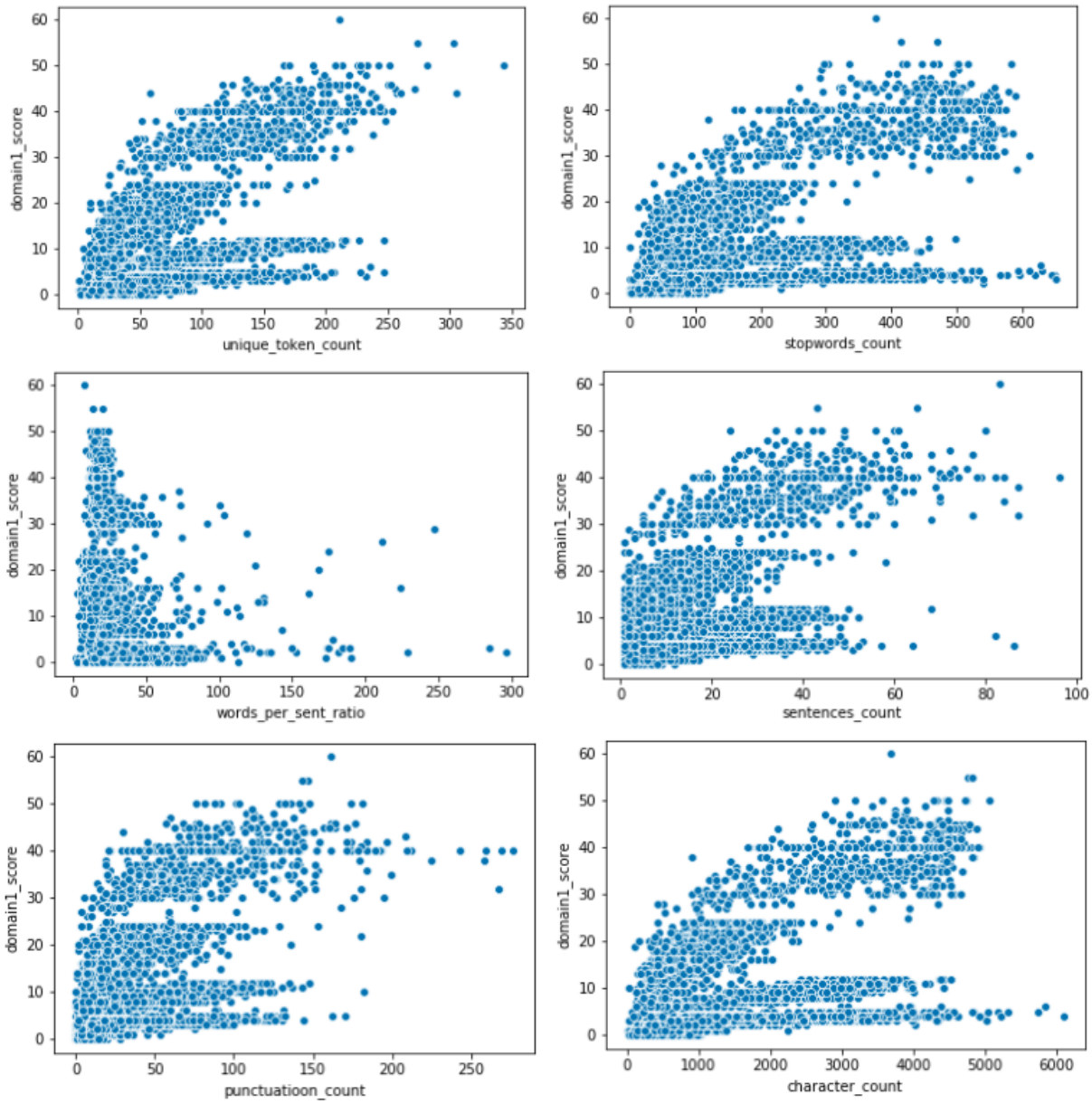


Рис. 3.4 – Вплив різних ознак на оцінку.

3.3 Векторизація текстових даних

Для векторизації тексту був використаний метод TF-IDF векторизації. Цей метод був обраний бо він надає менші ваги словам, що часто повторюються та навпаки більші ваги унікальним словам. В якості одного токена протягом векторизації використовувались 3-грами (послідовності з 3 слів). Протягом розробки були також випробувані інші варіанти векторизації,

наприклад, стандартна модель “торби слів” та інші розміри н-грам, однак всі вони дали дещо гірші результати на перехресній перевірці.

Зрозуміло, що використання повнорозмірного текстового вектора тільки покращує продуктивність моделі, проте, у нашому випадку такий вектор буде дуже великим, що сильно збільшує час навчання моделей. Тому було прийнято рішення скористатися методом головних компонент та зменшити розмірність текстових векторів до 50 (50 головних компонент). Така кількість компонент була знайдена емпіричним шляхом (при такому значенні час потрібний на навчання є придатним).

3.4 Відбір та нормалізація числових ознак

Заради пришвидшення навчання та покращення якості роботи моделі відбір числових ознак. Відбір був організований на основі статистики хі-квадрат. Було обраховано хі-квадрат статистику для кожної ознаки та обрано 15, що найбільш сильно впливають на цільову змінну. Важливість ознак потім зображувалася за спаданням, для кожного типу есе окремо, використовуючи можливості алгоритму випадкового лісу (рис. 3.5). Кожен тип есе розглядався окремо і під час відбору ознак. Нормалізація числових ознак була проведена методом стандартизованої оцінки (z-оцінки).

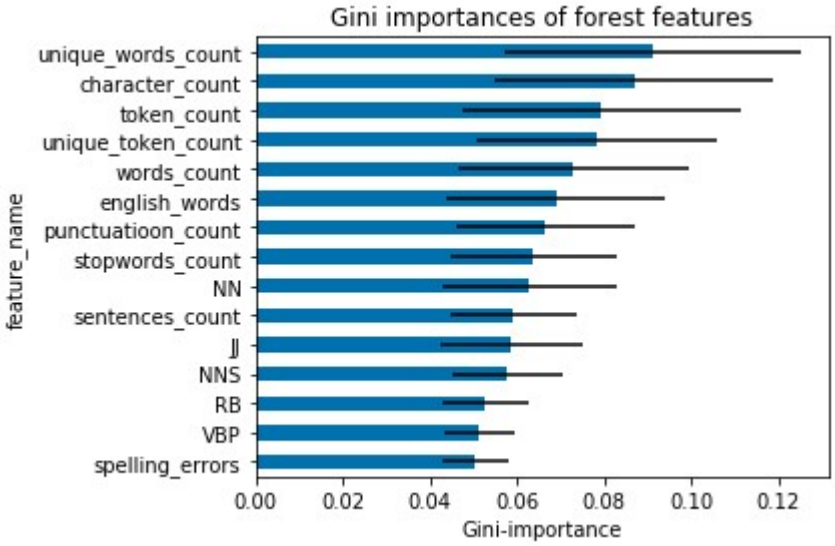


Рис. 3.5 – Важливість ознак першого типу творів.

3.5 Схема валідації моделі

На нових даних модель перевірялась два рази. Перший раз методом перехресної перевірки для вибору найкращої моделі та її гіперпараметрів, запобігання перенавчанню. Другий раз на тестовій вибірці для отримання фінальних результатів на даних, які модель побачила вперше. Обидві перевірки робилися окремо для кожного типу творів. Якість передбачень моделі оцінювалась із використанням коефіцієнта капи Коена.

3.6 Обрана модель

3.6.1 Стекінг

Для типів есе 1-7 була використана модель мультикласової класифікації на основі стекінгу. Стекінг [16] – це один з найпопулярніших методів ансамблювання алгоритмів, тобто використання декількох алгоритмів машинного навчання одночасно для покращення якості передбачень. В стекінгу ми спочатку навчаємо базові алгоритми (наприклад логістичну регресію та метод опорних векторів) та робимо передбачення на їх основі. Вектори передбачень базових алгоритмів ми потім використовуємо в якості ознак (також називають метаознаками) для так званого метаалгоритму. Передбачення ж метаалгоритму вже є нашою фінальною відповіддю. Для запобігання перенавчання, метаознаки (передбачення базових алгоритмів) отримуються тільки шляхом перехресної перевірки (рис 3.6). Очевидно, що базові алгоритми повинні бути якомога різноманітнішими, щоб кожен з них міг уловити якусь свою закономірність в даних та покращити результати фінальної моделі.

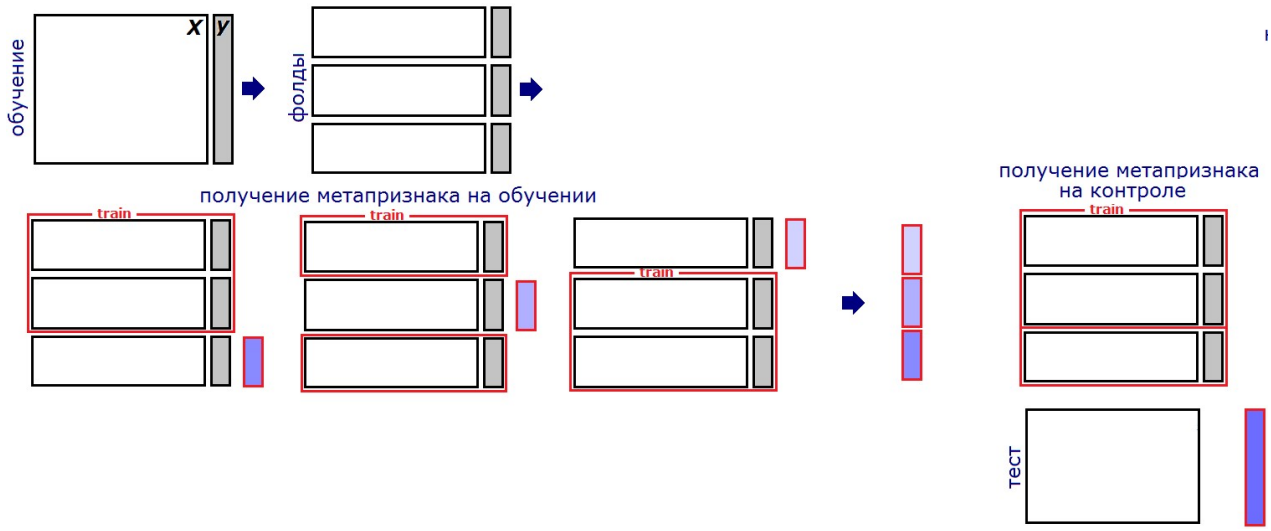


Рис 3.6 – Отримання метаознак при класичному стекінгу.

У нашому випадку у якості базових алгоритмів були використані: логістична регресія з регуляризаційним параметром L2, алгоритми випадкового та надзвичайно випадкового лісу з 70-а деревами рішень. Гіперпараметри міри регуляризації та кількості дерев були підібрані методом пошуку по ґратці (або варіація параметрів), який просто робить повний перебір по заданій вручну підмножині простору гіперпараметрів. Як метаалгоритм була використана звичайна логістична регресія. Метаознаки були отримані використовуючи перехресну перевірку з п’ятьма взаємодоповнюваними під-вибірками. Така сама система перевірки була використана і для оцінки якості роботи метаалгоритму.

3.6.2 Бегінг

У свою чергу, використані алгоритми випадкового та надзвичайно випадкового лісу також є ансамблями, вже на основі *бегінгу* (англ. bootstrap aggregating, bagging) та *методу випадкових підпросторів* (англ. random subspace method) з базовими алгоритмами у вигляді дерев рішень. Бегінг [16] заснований на тому, що з навчальної вибірки довжини l формуються різні навчальні підвибірки тієї ж довжини l за допомогою *бутстрепа* - випадкового вибору з поверненнями. При цьому деякі об'єкти потрапляють в підвибірку по кілька разів, деякі - ні разу. Можна показати, що частка

об'єктів, що опинилися в кожній підвибірці, прагне до $1 - e^{(-1)} \approx 0.632$ при $l \to inf$. Базові алгоритми, які пройшли навчання на підвибірках, об'єднуються в композицію за допомогою простого голосування. Ефективність бегінгу пояснюється двома обставинами. По-перше, завдяки різним базовим алгоритмам, їх помилки взаємно компенсуються при голосуванні. По-друге, об'єкт-викид може не потрапити до деяких навчальних підвибірок. Тоді алгоритм, побудований на такій підвибірці, може виявитися навіть точнішим за побудований на всьому навчальному наборі. Бегінг особливо ефективний на малих навчальних наборах, коли виключення навіть невеликої долі прикладів призводить до побудови суттєво різних алгоритмів.

3.6.3 Метод випадкових підпросторів та модель випадкового лісу

У методі випадкових підпросторів (random subspace method, RSM) базові алгоритми навчаються на різних підмножинах ознак, які також виділяються випадковим чином [17]. Цей метод використовують в задачах з великим числом ознак і відносно невеликим числом прикладів, а також при наявності надлишкових неінформативних ознак. У цих випадках алгоритми, побудовані на частині простору ознак, можуть володіти кращою здатністю до узагальнення в порівнянні з алгоритмами, побудованими за всіма ознаками.

Випадковий ліс використовує одночасно метод випадкових підпросторів та бегінг. Розбиття об'єктів в вершині випадкового лісу шукається серед випадкового підмножини ознак, а навчання кожного дерева в композиції відбувається на вибірці, отриманій за допомогою операції бутстрепу. Алгоритм надзвичайно випадкового лісу працює за схожими принципами, проте, навчання відбувається на всій вибірці та вибір найкращої ознаки для створення правила з випадкової підмножини ознак, відбувається випадковим чином.

3.6.4 Порівняння з іншими моделями

Нижче наведена таблиця порівняння різних реалізацій класифікатора з обраним варіантом (бралося середнє значення метрики якості по всім типам есе).

	QWK перехресна перевірка	QWK тестова вибірка
Логістична регресія	0.74	0.67
Дерево рішень	0.66	0.57
Випадковий ліс	0.74	0.67
Надзв. випадковий ліс	0.72	0.68
Ада Бустінг	0.63	0.55
Мій варіант стекінгу	0.75	0.70

Табл. 3.1– Результати різних моделей класифікації

Дивлячись на таблицю 3.1 стає зрозуміло, що найкраще для вирішення проблеми підходять логістична регресія, випадковий та надзвичайно випадковий ліс, ансамбль цих моделей, очікувано, тільки покращив результат.

3.6.5 Регресія хребта у якості класифікатора

Оскільки 8-й тип творів сильно відрізнялася від інших для нього була обрана окрема модель визначення оцінки. Як ми можемо побачити на рисунку 3.3, в цій підгрупі дуже великий діапазон можливих оцінок (17 категорій). Спочатку були випробувані моделі мультикласової класифікації, однак, всі вони давали посередні результати на перехресній перевірці, близько 0.4 коефіцієнта “каппа Коена” та були мною відкинуті. В результаті для цієї підгрупи було вирішено використовувати моделі регресійного аналізу з подальшим округленням до найближчого натурального числа, а саме лінійну регресію з регуляризаційним параметром L2, відому як регресія хребта.

3.7 Метрична оцінка якості обраної моделі

Як вже зазначалося для метричної оцінки якості передбачень у роботі використовувався коефіцієнт капи Коена. Можна використати оригінальну статтю [18] для оцінки значень коефіцієнта: значення ≤ 0, вказує на відсутність згоди, 0,01–0,20 незначна згода, 0,21–0,40 посередня згода, 0,41–0,60 помірна згода, 0,61–0,80 істотна згода, а 0,81–1,00 - майже ідеальне узгодження. У нашому випадку маємо такі результати перехресної перевірки для кожної підгрупи творів відповідно:

1. 0.82 – майже ідеальне узгодження
2. 0.69 – істотна згода
3. 0.70 – істотна згода
4. 0.76 – істотна згода
5. 0.80 – істотна згода
6. 0.74 – істотна згода
7. 0.75 – істотна згода
8. 0.70 – істотна згода

Середній показник по усіх підгрупах: 0.75 – істотна згода.

Показники на тестовій вибірці трошки відрізняються (особливо друга підгрупа):

1. 0.74 – істотна згода
2. 0.50 – помірна згода
3. 0.64 – істотна згода
4. 0.63 – істотна згода
5. 0.78 – істотна згода
6. 0.75 – істотна згода
7. 0.68 – істотна згода
8. 0.60 – істотна згода

Середній показник по усіх підгрупах: 0.70 – істотна згода.

Висновки до розділу 3

В даному розділі був описаний набір даних, що використовувався для навчання та основні етапи розробки моделі. За допомогою порівняльного аналізу визначено, ансамбль з моделей логістичної регресії, випадкового та надзвичайно випадкового лісу, у якості базових алгоритмів та логістичної регресії у якості мета-алгоритму дала найкращі показники обраних метрик якості та претендує на базову модель для підгруп 1-7. Для підгрупи 8 була обрана модель регресії хребта.

РОЗДІЛ 4
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Мова програмування та використані бібліотеки

У проекті була використана мова програмування Python. Python [19] – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією, розроблена в 1990 році Гвідо ван Россумом. Ця мова була обрана через простоту синтаксису, велику кількість навчальних матеріалів у відкритому доступі та наявність основних бібліотек машинного навчання.

Всі стандартні маніпуляції з даними (видалення колонок з пропусками, створення тестової вибірки, відображення даних у вигляді таблиць, та інші) були виконані з використанням бібліотеки Pandas. Pandas [20] – програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами. Pandas є вільним програмним забезпеченням, що випускається за ліцензією BSD. Бібліотека була обрана через її широкі можливості та оптимізовану продуктивність (критичні ланцюжки коду написано мовами Cython та C).

Візуалізація даних була виконана за допомогою бібліотеки Seaborn. Seaborn [21] – бібліотека візуалізації даних Python, заснована на matplotlib. Вона забезпечує інтерфейс високого рівня для відображення привабливої та інформативної статистичної графіки. Я обрав цю бібліотеку через спрощений інтерфейс порівняно зі стандартним matplotlib.

В процесі попередньої обробки тексту використовувалися бібліотеки NLTK та Spacy. NaturalLanguageToolkit (NLTK) [22] – це набір бібліотек і програм для символьної та статистичної обробки природних мов, написаної мовою програмування Python. Її розробили Стівен Берд й Едвард Лопер.

NLTK визначає інфраструктуру, яку можна використати для побудови програм NLP у Python. Вона надає базові класи для представлення даних, що мають відношення до обробки природної мови; стандартні інтерфейси для виконання таких завдань: анотування частин мови, синтаксичний розбір і класифікація тексту; і стандартні реалізації для кожного завдання, які можуть бути об'єднані для вирішення складних завдань. Удосконаленою, простішою у використанні та пришвидшено версією NLTK є бібліотека spaCy. Для пришвидшення роботи вона як і Pandas використовує Cython. Однак, повністю перейти лише на неї поки що складно бо вона набагато гірше документована.

Напевно найбільше під-час розробки цього дипломного проекту використовувалася бібліотека scikit-learn. Саме з її допомогою виконуються всі математичні перетворення, описуються та навчаються власне самі моделі. Scikit-learn [23] – це безкоштовна бібліотека машинного навчання для мови програмування Python. Вона містить майже всі сучасні алгоритми класифікації, регресії та кластеризації, чудово взаємодіє з науковими бібліотеками Python NumPy та SciPy.

4.2 Розроблені програмні модулі

Протягом реалізації дипломної роботи були розроблені два програмних модулі. Перший для аналізу даних, вибору та тренування моделей класифікації та трансформерів даних (класи в бібліотеці scikit-learn що використовують метод fit_transform). Другий модуль був розроблений у якості застосунку користувача.

Протягом розробки першого модуля використовувалася інтерактивна оболонка Ipython. Ipython [24] – інтерактивна оболонка мови програмування Python, яка поєднує можливості інтерактивної консолі Python і командної оболонки Unix, надає гнучкі засоби зневадження, редагування коду і візуалізації даних, інтроспекцію типів, додатковий shell синтаксис, підсвічування коду і помилок, tab-автодоповнення та іншу функціональність.

IPython активно використовується в науковому середовищі для розробки, обробки даних і інтерактивного виконання застосунків, пов'язаних з бібліотеками numpy, matplotlib, sympy і scipy. Зручність використання такої оболонки під час аналізу даних полягає у тому, що вона дозволяє виконувати частини коду (блоки) незалежно один від одного (рис 4.1). Це особливо важливо коли мова йде про реалізацію моделей машинного навчання, так як один блок коду може виконуватися лічені секунди, інший же – десятки годин. Наприклад, у випадку цієї дипломної роботи, блок коду, що відповідає за попередню обробку даних виконується приблизно годину, тоді як тренування моделей всього 10 хвилин, отже, IPython дозволяє нам виконати обробку один раз, а потім лише експериментувати з навчанням різних моделей. Більш того, використовуючи Jupyter Notebook (веб-середовище для роботи з IPython) є можливість комбінувати блоки коду з блоками текстів, картинок або формул, ця особливість дуже цінується в науковому середовищі.

```
[2]: # downloading the data
essays = pd.read_csv('train_set.csv')
essays_test= pd.read_csv('test_set.csv')
```

```
[3]: essays.head()
```

[3]:

	essay_set	essay	domain1_score
0	1	Dear local newspaper, I think effects computer...	8.0
1	1	Dear @CAPS1 @CAPS2, I believe that using compu...	9.0
2	1	Dear, @CAPS1 @CAPS2 @CAPS3 More and more peopl...	7.0
3	1	Dear Local Newspaper, @CAPS1 I have found that...	10.0
4	1	Dear @LOCATION1, I know having computers has a...	8.0

```
[4]: print(essays.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11680 entries, 0 to 11679
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   essay_set      11680 non-null  int64
 1   essay          11680 non-null  object
 2   domain1_score  11680 non-null  float64
dtypes: float64(1), int64(1), object(1)
memory usage: 273.9+ KB
```

Рис. 4.1:Приклад блоків IPython.

Після проведення аналізу даних та вибору найкращої моделі класифікатора, ця модель та всі трансформери (трансформер вибору найкращих ознак та трансформер зменшення розмірності) були збережені у форматі pickle (звичайна послідовність байтів) та передані у другий модуль програм.

Другий модуль був написаний на звичайному Python та реалізує програму-додаток. Користувач вводить есе у файл під назвою *essay_example.txt,* програма проводить трансформацію текстових даних та робить передбачення на основі моделей збережених у першому модулі. Зміни у першому модулі автоматично призводять до змін у другому. Нижче можна побачити приклад роботи програми.

```
michael@michael-Vostro-14-5459:~/Desktop/thesiswork/code$ cat essay_example.txt
Dear people, Computers have been here for not a very long time and everyday more people buy computers and everyday people use them. As they are used f
or entertainment, education, and talking with people faraway, they have had a negative effect on people. Let me explain. First of all, as people use c
omputers in the beginning they enjoy it. They go on the @CAPS1, play games, and use it for work. But as they keep using it, they start to get addicted
. People are addicted to computers because of the games and being able to go on the @CAPS1. In fact, @PERCENT1 of all the people in this @CAPS3 who ow
ns a @CAPS8, is addicted to computers. That percentage is expected to grow to @PERCENT2 in the next @NUM1 years. Addiction to computers lead to seriou
s issues. People have died because they spent too much time on the @CAPS8 without eating and sleeping. In another newspaper not long ago, it stated th
at a student attending @ORGANIZATION1, was dropped out for doing so poorly during school. His parent found out that he was addicted to this game "@CAP
S3 of @CAPS4." This game is right now the most played game on the @CAPS8 in this @CAPS3. Studies state that @CAPS8 addiction is almost as bad as smoki
ng addiction. My second reason why @CAPS8 has a negative effect on people is because of online bullying. People chat on @CAPS5, @CAPS6, @CAPS7, and em
ail and people are bullied. A friend of mine recently was emailed with words that hurt his feeling. Online bullying is worse that inteacting bullying.
 That's because @CAPS9 you say something on the @CAPS8, people don't know who you are. @PERSON1, a proffesor who recently did studies of online bullyi
ng says, "Online bullying has become more severe than contact to contact bullying. People are threatened more and hear more bad words than interacting
." The online bullying and harassment is going to get worse @CAPS9 people don't stop. Also people can get arrested for what they write online. My fina
l reason why computers have a negative effect on people is that people spend too much time on computers and don't exercise @CAPS8 contribute about @PE
RCENT3 to overweight people. On the @CAPS8, everything is much easier. Instead of mailing, you could e-mail. You can listen to music instead of walkin
g up to the radio and turning it on. You could buy tickets, shop, talk with friends, and have fun just by sitting on a @CAPS8. But this is why people
become overweight and obese. @PERCENT4 of the people who use computers are obese or overweight. Peole spend less time interacting with friends and fam
ily and enjoying nature. In conclusion, computers have a negative effect on people and parents should control their child's time on the computers. Jus
t because of a @CAPS8, many issues rise. So @CAPS9 you think you are addicted, try to make a time schedule for your whole day. Use the @CAPS8 for an h
our a time and take rests before using it again. Addiction, online bullying, and becoming overweight is not the only problems of a @CAPS8, there are m
any more.
michael@michael-Vostro-14-5459:~/Desktop/thesiswork/code$ python cliapp.py
Программа оценки эссе
Начата оценка эссе
Выберите тип изложения
Введите 1 если это эссе на свободную тему
Введите 2 если это изложение по прочитанному произведению
1
Предполагаемая оценка данного эссе по 12-ти бальной шкале: 10.0
```

Рис. 4.2: Приклад реалізації у вигляді консольного додатку.

Існує також можливість реалізації у вигляді веб-додатку (рис. 4.3).

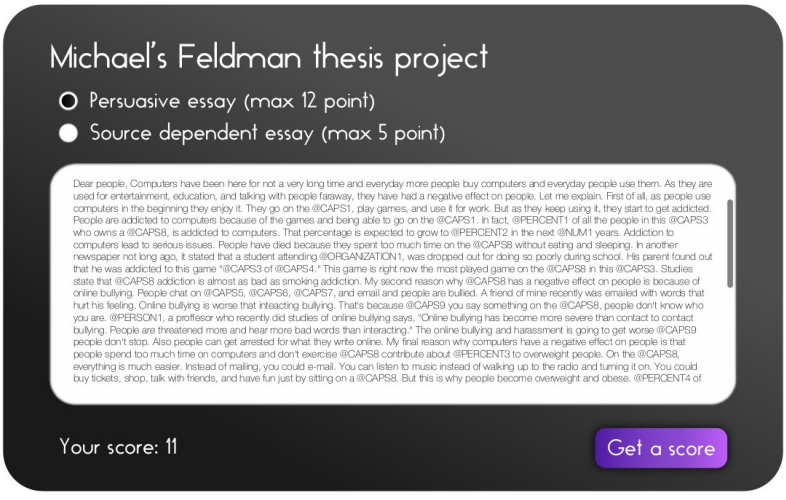


Рис. 4.3:Приклад реалізації у вигляді веб-додатку.

Висновки до розділу 4

В даному розділі були обумовлені та описані всі бібліотеки та мова, що були використані протягом розробки проекту. Були також продемонстровані середа розробки та вигляд програми-додатка.

ВИСНОВКИ

В даній дипломній роботі було розроблено та реалізовано систему автоматичного оцінювання невеликих текстових творів, написаних англійською мовою. Оцінювання відбувалося шляхом статистичної класифікації, використовуючи методи та моделі машинного навчання з учителем. Програмний продукт виконаний на мові програмування Python, за допомогою бібліотек Pandas, Seaborn, NLTK, Spacy та Scikit-learn.

У першому розділі роботи було проведено аналіз існуючих методів класифікації текстових документів. Були розглянуті, як варіанти на основі простого створення правил вручну, так і на основі машинного навчання з учителем та без нього. За підсумком аналізу існуючих методів було обґрунтовано використання саме методів машинного навчання для вирішення завдань пов'язаних з текстовою класифікацією.

В другому розділі були описані основні моделі, методи та етапи класифікації текстових документів з використанням машинного навчання. Зокрема були проаналізовані: методи попередньої обробки тексту та представлення його у векторному вигляді; методи конструювання, відбору та нормалізації ознак; проблема перенавчання та способи її вирішення; популярні моделі машинного навчання; способи валідації та оцінки якості роботи моделей.

В третьому розділі була обрана, описана та обґрунтована математична модель алгоритму класифікації та сам алгоритм. Був описаний набір даних, що використовувався для навчання, визначена метрика оцінки якості роботи моделі, детально проаналізовані та проілюстровані всі обрані етапи і методи. Була проведена метрична оцінка якості роботи обраної моделі, як за допомогою перехресної перевірки, так і за допомогою тестової вибірки. За результатами усередненої по всім типам творів перехресної перевірки був отриманий результат 0.75 метрики капи Коена, що свідчить про дуже істотну згоду між оцінками даними сертифікованими вчителями та системою

розробленою в даній дипломній роботі. Однак, результати сильно відрізняються залежно від типу есе (найкраще система працює з есе на вільну тему). На першому типі творів був отриманий результат перехресної перевірки в 0.82, що свідчить про майже ідеальну згоду в оцінюванні між вчителями та системою.

В четвертому розділі була описана реалізація з точки зору програмного забезпечення. Було обґрунтоване використання конкретної мови, бібліотек та середи розробки. Були запропоновані два варіанти інтерфейсу користувача для взаємодії з системою.

За результатами дослідження ефективності загальна ефективність розробленої системи на обраному наборі даних склала 0.75 метрики капи Коена. У вільному доступі мені не вдалося знайти кращого вирішення цієї задачі на цьому наборі даних (всі вирішення показували на 2-5% гірші результати), використовуючи методи стандартного машинного навчання. Були знайдені рішення на основі глибокого навчання (нейронні мережі), які перевершують результати розробленої системи на 1-3%. Однак, час потрібний на розробку, навчання та підтримку таких систем є в багато разів більшим, а вимоги до апаратури, необхідної для розробки десяти або більше гігабайтних систем на основі глибокого навчання, взагалі неможливо порівняти з системою вагою в 100 магабайт, що була розроблена в цій роботі.

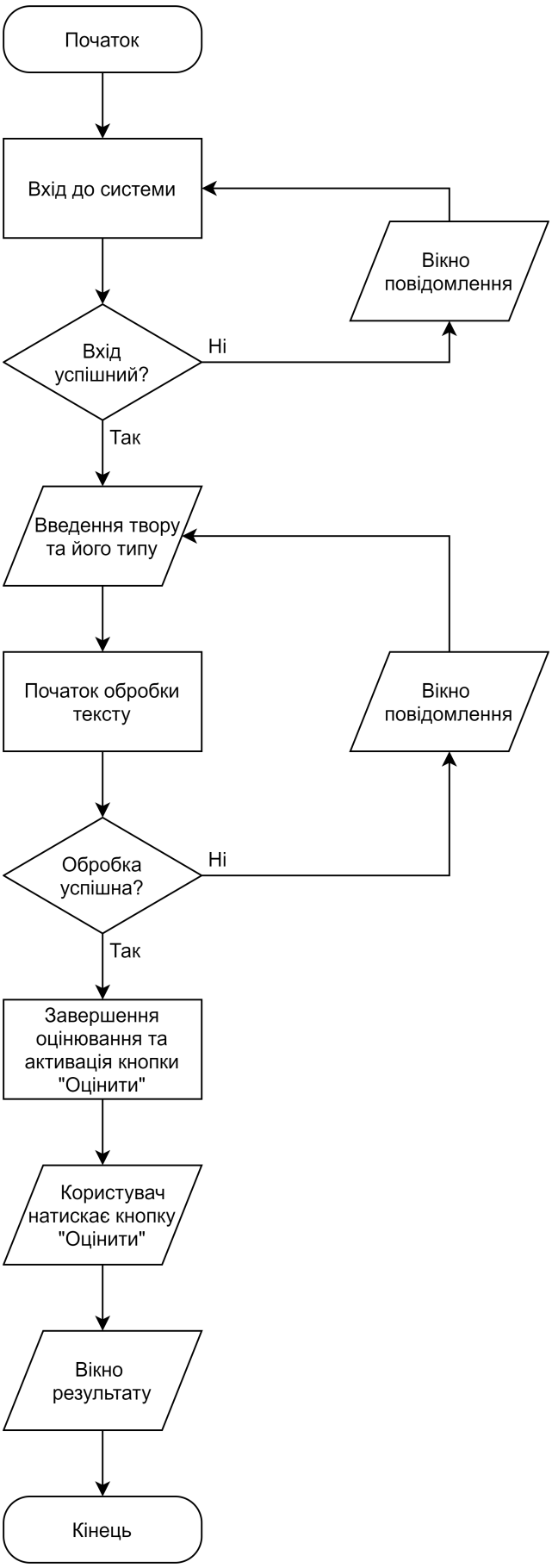
Новизна розробки полягає в широкому використанні ансамблевих методів машинного навчання. Всі задачі поставлені перед початком виконання роботи були виконані.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

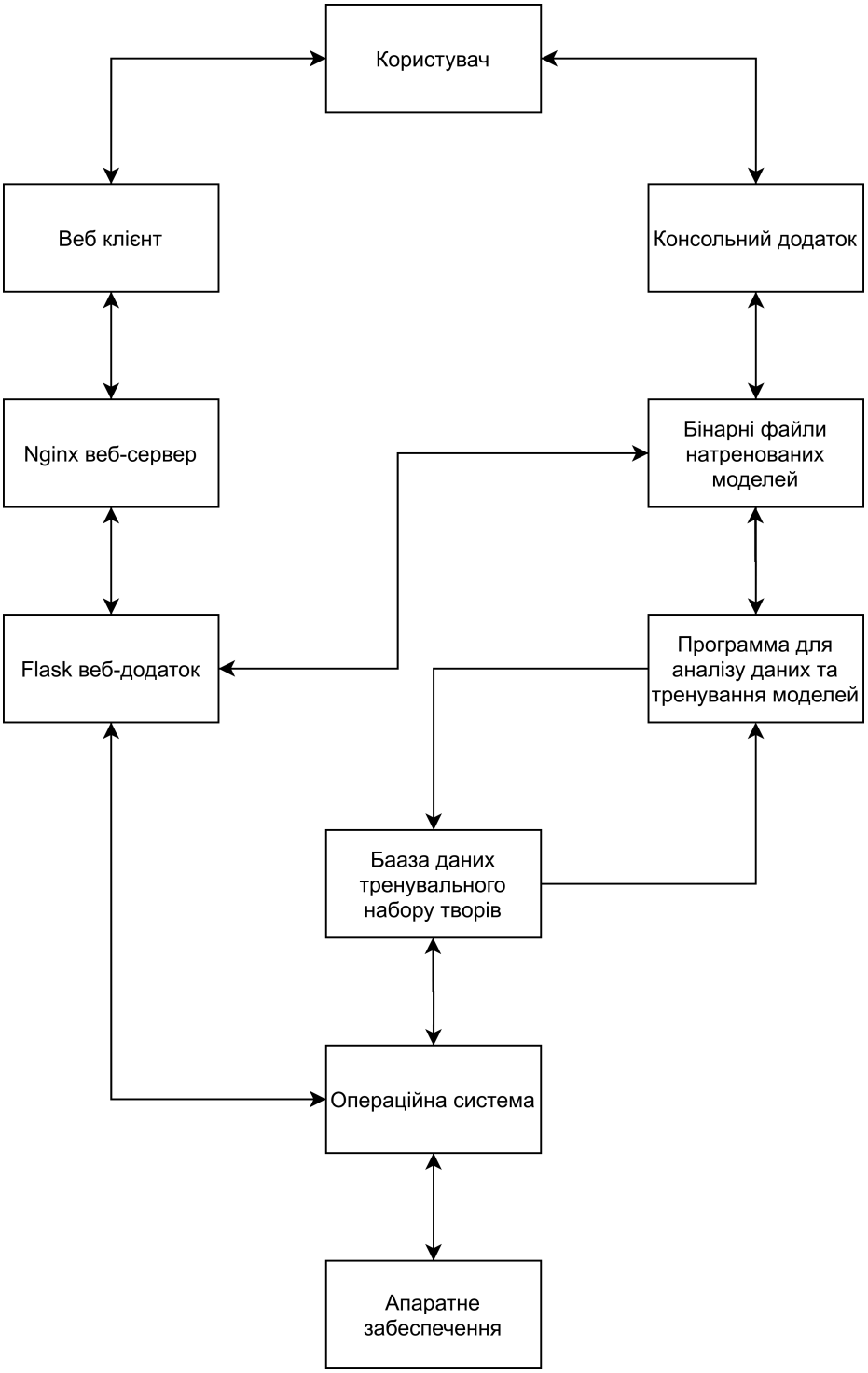
1. Mitchell T. Machine Learning / Tom Mitchell., 1997.
2. Aurélien G. The Machine Learning Landscape / Géron Aurélien // Hands-On Machine Learning with Scikit-Learn and Tensorflow / Géron Aurélien., 2017. – (1 edition). – С. 4–6.
3. Burkov A. Supervised Learning / Andriy Burkov // The Hundred-Page Machine Learning Book / Andriy Burkov., 2019. – (1 edition). – С. 3–10.
4. Tokenization [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Lexical_analysis#Tokenization.
5. NLP Essentials: Removing Stopwords and Performing Text Normalization using NLTK and spaCy in Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/.
6. Stemming and Lemmatization [Електронний ресурс]. – 2009. – Режим доступу до ресурсу: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.
7. Part-of-speech tagging [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Part-of-speech_tagging.
8. Named-entity recognition [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Named-entity_recognition.
9. Feature Engineering and Selection [Електронний ресурс] – Режим доступу до ресурсу: https://mlcourse.ai/articles/topic6-features/.
10. TF-IDF [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/TF-IDF.
11. Word2vec [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Word2vec.

12. Unsupervised Learning [Електронний ресурс] – Режим доступу до ресурсу: https://mlcourse.ai/articles/topic7-unsupervised/.
13. Classification using Decision Trees and k Nearest Neighbors [Електронний ресурс] – Режим доступу до ресурсу: https://mlcourse.ai/articles/topic3-dt-knn/.
14. Burkov A. Fundamental Algorithms / Andriy Burkov // The Hundred-Page Machine Learning Book / Andriy Burkov., 2019. – (1 edition). – С. 3–10.
15. Quadratic Weighted Cohen's kappa [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/aroraaman/quadratic-kappa-metric-explained-in-5-simple-steps.
16. Стекинг (Stacking) и блендинг (Blending) [Електронний ресурс] – Режим доступу до ресурсу: https://goo-gl.su/4unhI
17. Skurichina M., Duin R. P. W. Limited bagging, boosting and the random subspace method for linear classifiers // Pattern Analysis & Applications. 2002. С. 121–135.
18. Cohen J. A Coefficient of Agreement for Nominal Scales / Cohen Jacob – Ney York city, 1960.
19. Python [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Python.
20. Pandas [Електронний ресурс] – Режим доступу до ресурсу: http://pandas.pydata.org/.
21. Seaborn python library [Електронний ресурс] – Режим доступу до ресурсу: https://seaborn.pydata.org/.
22. First Steps In NLP NLTK [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/articles/first-steps-in-nlp-nltk/.
23. Scikit-learn [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Scikit-learn.
24. IPython [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/IPython.

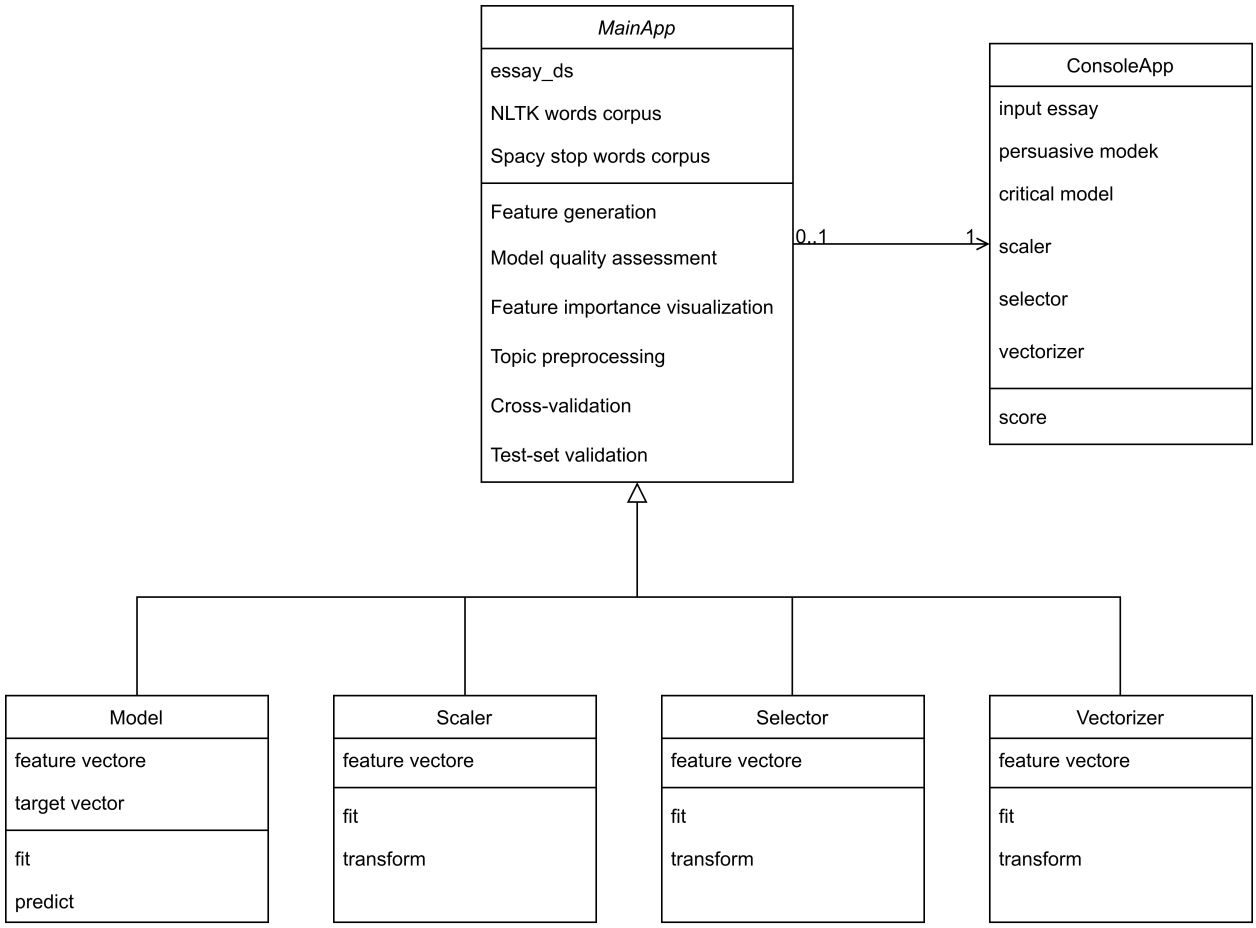
ДОДАТКИ



					ДП.6427.04.000 Д1		
Зм.		№ документа	Підп.	Дата			
					Система автоматичного оцінювання творів Додаток А	Літ.	Аркуш
						Т	1 / 1
Н.конт		Симоненко В.П.				НТУУ «КПІ імені Ігора Сікорського», ФІОТ Група ІО - 64	
Затв.							



					ДП.6427.05.000 Д2		
Зм.		№ документа	Підп.	Дата			
					Система автоматичного оцінювання творів Додаток Б	Літ.	Аркуш
						Т	1
Н.конт		Симоненко В.П.					1
Затв.						НТУУ «КПІ імені Ігоря Сікорського», ФІОТ Група ІО - 64	



Зм.		№ документа	Підп.	Дата	ДП.6427.06.000 ДЗ		
					Система автоматичного оцінювання творів	Літ.	Аркуш 1 / 1
Н.конт		Симоненко В.П.					НТУУ «КПІ імені Ігоря Сікорського», ФІОТ
Затв.					Додаток В		Група ІО - 64

Лістінг Програми

1) Аналіз даних та тренування моделей

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.sparse import hstack
import spacy
import en_core_web_sm
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
import nltk
from catboost import CatBoostClassifier, CatBoostRegressor
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif, f_regression
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransformer
from sklearn.model_selection import cross_val_score, GridSearchCV, cross_val_predict, KFold,
StratifiedKFold, train_test_split
from sklearn.metrics import classification_report, roc_auc_score, cohen_kappa_score, make_scorer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LinearRegression, ElasticNet
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor, ExtraTreesClassifier
from sklearn.feature_selection import chi2
from sklearn.svm import SVC, LinearSVC
from tqdm import tqdm_notebook
from pathlib import Path
words_english = set(nltk.corpus.words.words())
correct_words_set = set(nltk.corpus.brown.words())
lematizator = en_core_web_sm.load()
plt.style.use('seaborn-colorblind')

# downloading the data
RANDOM_STATE = 47
essays = pd.read_excel('~/Desktop/thesiswork/code/training_set_rel3.xlsx')
essays = essays[['essay_set','essay', 'domain1_score']].dropna()
essays.head()
print(essays.info())

#Topics distribution
sns.countplot(essays['essay_set'])

#Scors distribution in each topic
topic_number = 0
fig, ax = plt.subplots(4,2, figsize=(9,9), sharey=False)
for i in range(4):
   for j in range(2):
      topic_number += 1
      essays[essays['essay_set']==
topic_number].groupby('domain1_score')['essay_set'].agg('count').plot.bar(ax=ax[i, j], rot=0)
      ax[i,j].set_title('Topic %i' % topic_number)
```

```
ax[3,0].locator_params(nbins=10)
ax[3,1].locator_params(nbins=10)
plt.suptitle('Histograms of essay scores')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

#List of all parts of speech
list_of_all_tags=['LS', 'TO', 'VBN', "''", 'WP', 'UH', 'VBG', 'JJ', 'VBZ', '--', 'VBP', 'NN', 'DT', 'PRP', ':', 'WP$', 'NNPS',
'PRP$', 'WDT', '(', ')', '.', ',', '``', '$', 'RB', 'RBR', 'RBS', 'VBD', 'IN', 'FW', 'RP', 'JJR', 'JJS', 'PDT', 'MD', 'VB', 'WRB',
'NNP', 'EX', 'NNS', 'SYM', 'CC', 'CD', 'POS']

# Choosing essay topic, reseting indexes and lowering all words
essays = essays[(essays['essay_set']==4)]
essays['essay'] = essays['essay'].apply(lambda x : x.lower())
essays.reset_index(drop=True,inplace=True)

#Splitting essays by sentences and counting them
nltk_sentences_tokinized = []
for i in tqdm_notebook(essays['essay']):
   nltk_sentences_tokinized += [nltk.sent_tokenize(i)]
essays['essay_sentences'] = nltk_sentences_tokinized
essays['sentences_count'] = essays["essay_sentences"].apply(lambda x:len(x))

#Removing punctuation and counting words
nltk_words_tokinized = []
for i in tqdm_notebook(essays['essay']):
   nltk_words_tokinized += [[word for word in nltk.word_tokenize(i) if word not in punctuation]]
essays['essay_words'] = nltk_words_tokinized
essays['words_count'] = essays['essay_words'].apply(lambda x:len(x))

#Counting punctuation, spelling errors, charachters, words per sentence ratio, average sentence length
def words_filter(essays):
   punctuation_count = []
   misspelled_words_count = []
   for essay in tqdm_notebook(essays):
      pc_temp=0
      mw_temp=0
      for word in nltk.word_tokenize(essay):
         if word.lower() not in punctuation:
            if word.lower() not in correct_words_set:
               mw_temp+=1
         else:
            pc_temp+=1
      punctuation_count.append(pc_temp)
      misspelled_words_count.append(mw_temp)
   return punctuation_count,misspelled_words_count
word_features = words_filter(essays['essay'])
essays['punctuatioon_count'] = word_features[0]
essays['spelling_errors'] = word_features[1]
essays['character_count'] = essays['essay'].apply(lambda essay:len(essay))
essays['words_per_sent_ratio'] = round(essays['words_count']/essays['sentences_count'])
essays['avg_sen_len'] = round(essays['character_count']/essays['sentences_count'])

# Counting only english words
english_words=[]
for listofwords in tqdm_notebook(essays['essay_words']):
```

```
    temp_english_word=0
    for word in listofwords:
        if word in words_english:
            temp_english_word+=1
    english_words.append(temp_english_word)
essays['english_words']=english_words

#Counting only non-stopwords and stopwords
nltk_words_tokinized_no_stop_words = []
stopwords_count = []
for tolinized_list in tqdm_notebook(essays['essay_words']):
    temp_stop_words_count = 0
    temp_nsw = []
    for word in tolinized_list:
        if word not in STOP_WORDS:
            temp_nsw.append(word)
        else:
            temp_stop_words_count+=1
    nltk_words_tokinized_no_stop_words.append(temp_nsw)
    stopwords_count.append(temp_stop_words_count)
essays['essay_words'] = nltk_words_tokinized_no_stop_words
essays['stopwords_count'] = stopwords_count

#Joining essay without stopwords
def words_joiner(essays):
    for line_number in tqdm_notebook(range(essays.index[0],essays.index[-1]+1)):
        essays['essay_words'][line_number] = ' '.join(essays['essay_words'][line_number])

#Tagging evry word with the corespondent part of speech
for i in list_of_all_tags:
    essays[i] = 0
for line_number in tqdm_notebook(range(essays.index[0],essays.index[-1]+1)):
    temp_pos_list = np.unique(np.array(nltk.pos_tag(essays['essay_words'][line_number]))[:,1],
return_counts=True)
    pos_list = list(zip(temp_pos_list[0],temp_pos_list[1]))
    for i in pos_list:
        essays[i[0]][line_number]=i[1]
essays.drop(columns=["''", '--', ':', '(', ')', '.', ',', '``', '$','essay','essay_sentences'],inplace=True,axis=1)
words_joiner(essays)

#Reducing every word to its grammatical root and counting absolute and unique number of those roots
lemmatized_essay_words = []
for essay in  tqdm_notebook(essays['essay_words']):
    lemmatized = [t.lemma_ for t in lematizator(essay)]
    lemmatized_essay_words.append(lemmatized)
essays['essay_words'] = lemmatized_essay_words
essays['token_count'] = essays['essay_words'].apply(lambda x: len(x))
essays['unique_token_count'] = essays['essay_words'].apply(lambda x: len(set(x)))
words_joiner(essays)

#Vectorization of text data
tv = TfidfVectorizer(ngram_range=(1, 5), max_features=100000)
essay_words_vectorized = tv.fit_transform(essays['essay_words'])
y_train = essays['domain1_score']

#All choosen ML models
```

```
lin_reg = LinearRegression(n_jobs=-1,normalize=True)
log_clf = LogisticRegression(C=20,n_jobs=-1,random_state = RANDOM_STATE)
catboost_reg = CatBoostRegressor(n_estimators=1000,random_state = RANDOM_STATE)
svm = LinearSVC(random_state = RANDOM_STATE)
random_forest = RandomForestClassifier(random_state = RANDOM_STATE)
catboost_clf=CatBoostClassifier(n_estimators=1000,random_state = RANDOM_STATE)
NBC=MultinomialNB()

#Numerical features selection
#sel_varience = VarianceThreshold(threshold=(.9 * (1 - .9)))
#selected_sel_varience=
sel_varience.fit_transform(essays.drop(['domain1_score','essay_words','essay_set'],axis=1))
sel_k_best = SelectKBest(chi2, k=15)
selected_sel_k_best=
sel_k_best.fit_transform(essays.drop(['domain1_score','essay_words','essay_set'],axis=1),y_train)

#Feature importances visualization
X=
essays[essays.drop(['domain1_score','essay_words','essay_set'],axis=1).columns[(sel_k_best.get_support())]]
y = y_train.astype(np.float64)
forest = ExtraTreesClassifier(n_estimators=250, random_state=26)
forest.fit(X, y)
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
# plot feature importances
features = pd.DataFrame({'feature_name': X.columns, 'importance': forest.feature_importances_, 'std': std})
features.sort_values('importance').plot.barh(x='feature_name', y='importance', xerr='std', legend=False)
plt.title('Gini importances of forest features')
plt.xlabel('Gini-importance')
plt.tight_layout()
plt.show()

#Performng numerical features scaling, stacking with vectorized text data and dimentionality reduction
pca = TruncatedSVD(n_components=15,random_state = RANDOM_STATE)
scaleer = StandardScaler(copy=True, with_mean=True, with_std=True)
numerical_scaled = scaleer.fit_transform(selected_sel_k_best)
essay_words_vectorized_reduced = pca.fit_transform(essay_words_vectorized)
X_reduced = np.concatenate((essay_words_vectorized_reduced,numerical_scaled),axis=1)
print(X_reduced.shape)

#Performing 5 fold cross-validation
model = log_clf
kappa_scorer = make_scorer(cohen_kappa_score)
splits = KFold(n_splits=5,random_state = RANDOM_STATE)
cv_scores_kappa_scorer = cross_val_score(model, X_reduced, y_train, cv=splits, scoring=kappa_scorer,
n_jobs=-1, verbose=True)
cv_scores_accuracy = cross_val_score(model, X_reduced, y_train, cv=splits, scoring='accuracy', n_jobs=-1,
verbose=True)

#Printing results
print('foldsaverageaccuracyscore:{},foldsaveragekappa
score:{}'.format(cv_scores_accuracy.mean(),cv_scores_kappa_scorer.mean()))
print('accuracy_scores:{}'.format(cv_scores_accuracy))
print('kappa_scores:{}'.format(cv_scores_kappa_scorer))

#Training the model on whole data than saving the model
model = LogisticRegression(C=10,n_jobs=-1,random_state = RANDOM_STATE)
```

```
model.fit(X_reduced, y_train)
pickle.dump(model,open('model.pkl','wb'))
```

2) Консольна програма застосунок

```
#importing all the nedded libs
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import spacy
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.feature_extraction.text import TfidfVectorizer
import en_core_web_sm
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
import nltk

#Starting the app + constants
print('Программа оценки эссе')
print('Начата оценка эссе')
RANDOM_STATE = 42
words_english = set(nltk.corpus.words.words())
correct_words_set = set(nltk.corpus.brown.words())
lematizator = en_core_web_sm.load()

#Getting the data
f = open('essay_example.txt','r')
essay_example = f.readlines()[0]
essay_example = essay_example.lower()
print('Выберите тип изложения')
print('Введите 1 если это эссе на свободную тему')
print('Введите 2 если это изложение по прочитанному произведению')
essay_type = input()

#counting first numerical features
essay_sentences = nltk.sent_tokenize(essay_example)
sentences_count = len(essay_sentences)
essay_words= [word for word in nltk.word_tokenize(essay_example) if word not in punctuation]
words_count =len(essay_words)
unique_words_count = len(set(essay_words))

if unique_words_count in range(10) or sentences_count in range(3):
    print('Изложение не подлежит проверке. Оценка - 0б')
    exit()

#punctuation counter and misspelled words counter
punctuation_counter=0
misspelledwords_counter=0
for word in nltk.word_tokenize(essay_example):
    if word.lower() not in punctuation:
        if word.lower() not in correct_words_set:
            misspelledwords_counter+=1
    else:
```

```
        punctuation_counter+=1
punctuatioon_count = punctuation_counter
spelling_errors = misspelledwords_counter

#other numerical features and english words counter
character_count = len(essay_example)
words_per_sent_ratio = round(words_count/sentences_count)
avg_sen_len = round(character_count/sentences_count)
temp_english_word=0
for word in essay_words:
    if word in words_english:
        temp_english_word+=1
english_words=temp_english_word

#stop words removal and counter
temp_stop_words_count = 0
temp_nsw = []
for word in essay_words:
    if word not in STOP_WORDS:
        temp_nsw.append(word)
    else:
        temp_stop_words_count+=1
essay_words = temp_nsw
stopwords_count = temp_stop_words_count

#parts of speach tagger and numerical features
list_of_pos_features = {'LS':0, 'TO':0, 'VBN':0, 'WP':0, 'UH':0, 'VBG':0, 'JJ':0, 'VBZ':0,
                        'VBP':0, 'NN':0, 'DT':0, 'PRP':0, 'WP$':0, 'NNPS':0, 'PRP$':0, 'WDT':0, 'RB':0, 'RBR':0,
                        'RBS':0, 'VBD':0, 'IN':0, 'FW':0, 'RP':0, 'JJR':0, 'JJS':0, 'PDT':0, 'MD':0, 'VB':0, 'WRB':0,
                        'NNP':0, 'EX':0, 'NNS':0, 'SYM':0, 'CC':0, 'CD':0, 'POS':0}
temp_pos_list = np.unique(np.array(nltk.pos_tag(essay_words))[:,1], return_counts=True)
pos_list = list(zip(temp_pos_list[0],temp_pos_list[1]))
for i in pos_list:
    if i[0] in list_of_pos_features.keys():
        list_of_pos_features[i[0]]=i[1]
essay_words = ' '.join(essay_words)
lemmatized_essay_words = [t.lemma_ for t in lematizator(essay_words)]
essay_words = lemmatized_essay_words
token_count = len(essay_words)
unique_token_count = len(set(essay_words))
essay_words = ' '.join(essay_words)

all_features=
np.array([sentences_count,words_count,unique_words_count,punctuatioon_count,spelling_errors,
                        character_count,words_per_sent_ratio,avg_sen_len,english_words,
                        stopwords_count,list_of_pos_features['LS'],
list_of_pos_features['TO'], list_of_pos_features['VBN'], list_of_pos_features['WP'], list_of_pos_features['UH'],
                        list_of_pos_features['VBG'], list_of_pos_features['JJ'], list_of_pos_features['VBZ'],
                        list_of_pos_features['VBP'], list_of_pos_features['NN'], list_of_pos_features['DT'],
                        list_of_pos_features['PRP'], list_of_pos_features['WP$'], list_of_pos_features['NNPS'],
                        list_of_pos_features['PRP$'],
list_of_pos_features['WDT'], list_of_pos_features['RB'], list_of_pos_features['RBR'],
                        list_of_pos_features['RBS'],
list_of_pos_features['VBD'], list_of_pos_features['IN'], list_of_pos_features['FW'], list_of_pos_features['RP'],
                        list_of_pos_features['JJR'],list_of_pos_features['JJS'], list_of_pos_features['PDT'],
                        list_of_pos_features['MD'], list_of_pos_features['VB'], list_of_pos_features['WRB'],
```

```
                list_of_pos_features['NNP'],
list_of_pos_features['EX'], list_of_pos_features['NNS'], list_of_pos_features['SYM'],
                list_of_pos_features['CC'],
list_of_pos_features['CD'], list_of_pos_features['POS'],token_count,unique_token_count])

#loading trained model and all the transformers
if essay_type == '1':
   model = np.load('pickles/persuasive_model.pkl',allow_pickle=True)
   tv = np.load('pickles/persuasive_vectorizer.pkl',allow_pickle=True)
   pca = np.load('pickles/persuasive_pca.pkl',allow_pickle=True)
   scaleer = np.load('pickles/persuasive_scaleer.pkl',allow_pickle=True)
   selector =  np.load('pickles/persuasive_selector.pkl',allow_pickle=True)
if essay_type == '2':
   model = np.load('pickles/critical_model.pkl',allow_pickle=True)
   tv = np.load('pickles/critical_vectorizer.pkl',allow_pickle=True)
   pca = np.load('pickles/critical_pca.pkl',allow_pickle=True)
   scaleer = np.load('pickles/critical_scaleer.pkl',allow_pickle=True)
   selector =  np.load('pickles/critical_selector.pkl',allow_pickle=True)

#transforming our essay
selected_sel_k_best = selector.transform(all_features.reshape(1, -1))
essay_words_vectorized = tv.transform([essay_words])
numerical_scaled = scaleer.transform(selected_sel_k_best)
essay_words_vectorized_reduced = pca.transform(essay_words_vectorized)
X_reduced = np.concatenate((essay_words_vectorized_reduced,numerical_scaled),axis=1)

#predicting the score!
prediction = model.predict(X_reduced[0].reshape(1,-1))[0]
if essay_type == '1':
   print(f'Предпологаемая оценка данного эссе по 12-ти бальной шкале: {prediction}')
if essay_type == '2':
   print(f'Предпологаемая оценка данного эссе по 5-ти бальной шкале: {prediction+1}')
```